Universidad
Carlos III de Madrid
www.uc3m.es

ESCUELA POLITÉCNICA SUPERIOR
INGENIERIA INDUSTRIAL
DEP. AUTOMÁTICA Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

KOMBOT. DISEÑO Y CONSTRUCCIÓN DE UN MINI ROBOT MÓVIL

Autor: ALBERTO MALDONADO SÁNCHEZ

Tutor: RAMÓN BARBER CASTAÑO

Noviembre 2015

Agradecimientos

La realización del presente proyecto ha sido posible gracias a la participación y ayuda de distintas personas.

En primer lugar me gustaría agradecer al profesor Juan González Gómez también conocido como “Obijuan” quien gracias a su pasión por la robótica me hizo interesarme por este tipo de robots.

Es importante para mí agradecer sobre todo al profesor Alberto Valero Gómez al que pude conocer por mediación de Juan, y que ha sido mi director y supervisor durante el comienzo del proyecto. Fue él quien me propuso participar en la construcción del robot junto a otros alumnos y quien me ha ido guiando en todo el proceso incluso cuando dejó la universidad.

Seguidamente quiero agradecer también la labor de Raúl Pérula Martínez quien tras la marcha de Alberto me permitió continuar con el proyecto proporcionándome todo lo necesario para la realización del mismo.

También quiero agradecer a Ramón Barber Castaño quien finalmente me ayudó a poder terminar el proyecto con éxito para así poder presentarlo.

Igualmente mi agradecimiento a los alumnos Sergio Vilches Expósito y Ruy García García; ya que sin su participación y colaboración en el robot, este proyecto no habría sido posible.

Finalmente agradecer a Marco Esteban Illescas por su ayuda con las impresoras 3D del laboratorio gracias a las cuales pude imprimir las piezas del chasis del robot.

Índice

Índice de figuras

1. Introducción

1.1. Idea y motivación

Nuestro proyecto consiste en el diseño y construcción de un pequeño robot móvil para desplazamiento sobre superficies planas, al que hemos denominado Kombot. La palabra Kom procede del esperanto komunumo y significa comunidad. Este proyecto tiene como objetivo desarrollar una herramienta fácil de usar; es decir, crear una plataforma móvil de bajo coste y que tenga como finalidad su uso en investigación y educación avanzada en robótica. El nombre del proyecto quiere indicar dos ideas fundamentales y que reflejan la intención por la cual ha sido creado:

- Toda la historia del desarrollo y los archivos que contiene serán de tipo abierto a la comunidad. Estarán a disposición de cualquiera que le interese para que todo el trabajo y esfuerzo puesto en la plataforma pueda ser utilizada por otras personas para que puedan adaptarlas a otro propósito diferente si así lo quieren.
- El objetivo de crear una plataforma de bajo coste tiene un doble propósito:

    - Reducir la barrera económica de entrada para la creación de un laboratorio de robótica móvil.
    - Permitir a los investigadores en robótica de enjambre probar sus algoritmos en hardware real.

La robótica en enjambre consiste en unir muchos robots simples de tal forma que hagan en conjunto trabajos más complicados que por sí solos no serían capaces de hacer. Un ejemplo de esto es el mapeado colectivo que consiste en colocar varios de estos robots en una zona concreta de manera que cada uno se mueve por una parte distinta de esa zona y van analizándola consiguiendo finalmente una imagen global de toda el área.

Nuestro proyecto surgió en el año 2012 con la idea de construir un robot que sirviese para investigación; así como enseñar a los alumnos de educación superior (Master) a construir y programar este tipo de robots.

Antes de la creación de nuestro robot, ya teníamos en ese momento como antecedentes previos dos proyectos similares al nuestro:

- Khepera
- E-puck

Khepera *(Figura 1)* es un pequeño robot móvil (5.5 cm) creado a mediados de los 90 en Suiza por Edo. Franzi, Francesco Mondada y André Guignard entre otros en el laboratorio LAMI del profesor Jean-Daniel Nicoud de la Escuela Politécnica Federal de Lausanne (EPFL). Ha servido durante años en la investigación y es utilizado en la actualidad por un gran número de universidades de todo el mundo. En la actualidad este robot ya va por la versión IV. Sin embargo en el momento en el que empezó a desarrollar nuestro proyecto, se estaba trabajando en la versión III de Khepera. Algunos detalles importantes de este robot son:

- Precio. Alrededor de los 2000€ el modelo más básico. La compra de diversos kits que lo complementen encarecen aún más su precio.
- Utiliza LabVIEW, una plataforma y entorno de desarrollo para el diseño de sistemas. Pertenece a National Instruments; lo que significa que es un software de pago.

E-puck *(Figura 2)* es otro robot móvil de ruedas diferencial diseñado originalmente para la educación micro-ingeniería por Michael Bonani y Francesco Mondada en el laboratorio ASL del Prof. Roland Siegwart en EPFL. E-puck ha tenido una gran aceptación, siendo utilizado en robótica colectiva, robótica evolutiva y en la robótica orientada al arte. Este robot fue desarrollado en 2005, siendo el 2007 la última versión de este modelo. Aquí podemos destacar:

- Precio. Se encuentra entre los 450€ y los 550€.
- Tanto el hardware como el software son de tipo open source.

El primero es un modelo bastante caro, mientras que E-puck tiene un coste más reducido gracias a que es de código abierto. Aun así son modelos que requieren una cantidad considerable de dinero para adquirirlos. Uno de nuestros principales objetivos será intentar mejorar este aspecto.

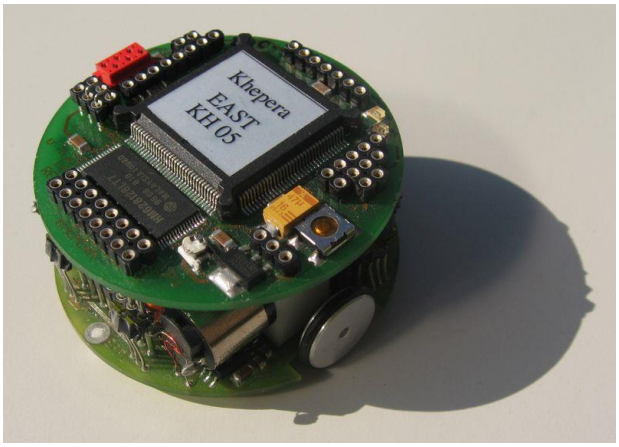


Figura 1: Robot Khepera

Figura 2: Robot E-puck

Es importante mencionar también que aunque la electrónica está creada desde cero; se encuentra basada en otro proyecto *(Figura 3)* de un profesor de la UC3M: Juan González Gómez.

La manera en que los programas se introducen en el microcontrolador está basada en dicho proyecto.

Figura 3: Robot SkyCube

Nuestra intención era desarrollar un robot móvil que sirviese para estos propósitos y que fuera enteramente desarrollado en la UC3M, pero permitiendo que cualquiera que estuviese interesado pueda adquirirlo.

El robot está hecho para que sea muy fácil de modificar. Se ha diseñado de tal forma que podamos incluirle más elementos:

- A nivel de hardware: A lo largo de todo el robot tenemos una serie de agujeros mediante los cuales podemos acoplarle cualquier pieza que queramos añadir. Bastará con imprimir la pieza deseada y unirla mediante unos tornillos al chasis.
- A nivel de electrónica: Permite añadirle más elementos gracias a que tiene un gran número de conectores (pines) en la placa con lo que podríamos añadirle más elementos uniéndolo mediante un cable.

Así es más fácil de reconfigurar, para adaptarlo a lo que queramos según la situación.

2. Objetivos

2.1. Objetivo del proyecto

El presente proyecto tiene como objetivo principal la creación de un pequeño robot móvil denominado Kombot.

Se quiere aumentar el interés por este tipo de proyectos para que cualquiera que esté interesado pueda contribuir en él y aportar sus ideas. Es un proyecto abierto para todo el mundo que siempre puede modificarse y mejorarse continuamente. Por eso estará a disposición de todo el mundo los archivos de programación y planos para que quien quiera pueda fabricar el suyo.

Veremos cada una de las partes de las que consta dicho robot (electrónica, mecánica e informática). Indicaremos en cada una su proceso de creación y los elementos necesarios para su construcción, tales como componentes o materiales necesarios para fabricarlo según el caso.

Se verán a su vez todos los distintos programas que se han empleado a lo largo del proyecto para poder construirlo e incluiremos una breve guía de ellos de cómo los hemos utilizado para que el lector tenga una noción de cómo usarlos en el caso de que le atraiga la idea.

Incluiremos también el presupuesto de todos los elementos que conforman parte de él y los proveedores de los que se han obtenido para facilitar la tarea en el caso de que se quiera construir otro modelo.

Buscaremos crear una estructura rígida y sencilla (compuesta por una placa y un chasis imprimible) y que a su vez sea más económica que los modelos existentes mencionados anteriormente.

3. Diseño electrónico

3.1. Introducción

El diseño de esta parte ha sido desarrollada por Sergio Vilches Expósito y engloba todos los componentes electrónicos de los que consta el robot y las distintas pruebas que se han realizado para su diseño (Esto último lo podemos encontrar en el apartado 7.2. del proyecto). Veremos los más relevantes a ellos divididos por categorías según su función.

También nos encontramos en esta parte a KiCad; el software utilizado para poder desarrollar esta parte.

Así mismo veremos también como se ha construido la placa impresa en la cual van colocados estos componentes.

Uno de los objetivos que se querían alcanzar en el diseño del robot es su pequeño tamaño. Queríamos optimizarlo de tal forma que fuese lo más pequeño posible sin limitar las funciones para las cuales ha sido diseñado. Por ello la disposición de todos los elementos está pensada para aprovechar al máximo el espacio del que disponemos.

Nuestro robot está basado en Arduino. Arduino es una plataforma de hardware libre que compuesto por una placa con un microcontrolador y un entorno de desarrollo propio. Engloba:

- Lenguaje de programación, que es similar a C++.
- Placa. Nosotros no usaremos la placa entera, lo que si utilizamos es el mismo chip microcontrolador (ATmega) de los que está compuesto.
- IDE (Integrated Development Environment) o Entorno de desarrollo integrado. Software usado para programar.

En el proyecto usamos para programarlo el entorno IDE de Arduino. Este lenguaje será también es que se utilice tanto para las pruebas de componentes del robot como la programación informática (un ejemplo de esto es la prueba de las Xbees).

Una decisión importante ha sido la de no optar por la compra de una placa completa de Arduino. Existen diversos modelos de placas basadas en Arduino y que aportan una serie de ventajas:

- Simplifica el proceso de trabajar con microcontroladores.
- Es multiplataforma, pudiendo utilizarse en Windows, Linux o Mac.
- Es compacta. Las placas incluyen microcontrolador, diversos pines digitales y analógicos, así como otros elementos hardware necesarios para su funcionamiento.

Sin embargo como hemos indicado, hemos decidido en nuestro caso no comprar una placa de Arduino. En nuestro caso optamos por comprar únicamente uno de los microcontroladores de los que utiliza normalmente. Las razones que nos han llevado a esta decisión son:

- Las placas de Arduino son menos flexibles, existen una serie de modelos con características fijas. En nuestro caso hemos diseñado una placa que se adapta mejor a nuestras necesidades.
- La placa de nuestro robot es específica para él. Una placa completa de Arduino supondría incorporarlo a nuestro caso y crear otro diseño que también lo incluyese. El volumen del robot sería mayor al tener dos placas: la Arduino y la que contuviese al resto de elementos.
- Nos evitamos una gran cantidad de cables que conectasen los distintos elementos al tener un diseño integrado en el que las pistas se encuentran en nuestra propia placa.
- Al utilizar el mismo microcontrolador podemos emplear las mismas herramientas, ya que son de tipo abierto. Esto nos permite acceder a una enorme cantidad de información en la que colabora una gran comunidad.

Aunque éste será un robot versátil, al cual podremos añadir y/o cambiar diferentes componentes en función de nuestras necesidades; en esta primera versión el robot estará compuesto por un sensor de infrarrojos y dos sensores de ultrasonidos. Así mismo también dispondremos de cuatro encoders que se encargarán de medir el movimiento de las ruedas motoras. Como actuadores contamos con dos pequeños motores así como de un servo. Esto lo veremos más adelante en sus apartados correspondientes.

3.2. Arquitectura del sistema

Estos componentes se pueden dividir en cinco clases distintas según la categoría a la que pertenecen:

- Microcontrolador central
- Gestión de energía
- Comunicaciones
- Sensores
- Actuadores

Existen otros elementos que también forman parte de la electrónica del robot, tales como resistencias, bobinas, condensadores, interruptores... Estos componentes no aparecen mencionados aquí por su menor relevancia, sin embargo son imprescindibles para el funcionamiento del robot. Por ello aparecerán reflejados en los planos finales del circuito así como en la lista de componentes del presupuesto.

3.2.1. Microcontrolador central

En esta primera parte hablaremos del microcontrolador, el cual corresponde con el cerebro del robot. Es la pieza central en la que se carga el programa y permite al robot realizar las distintas acciones.

Así mismo también nos centraremos en el gestor de arranque. Veremos de qué manera podemos conectar el ordenador al robot para así poder cargarle la información.

3.2.1.1. Microcontrolador

Como ya hemos indicado anteriormente, nuestro robot no lleva una placa Arduino completa; si no que empleamos un microcontrolador del mismo tipo que los que utilizan este tipo de placas. Con ello conseguimos:

- Abaratar los costes, puesto que no tenemos que comprar el bloque entero. Utilizamos ciertos componentes que mejor nos vienen para nuestro caso.
- Al utilizar el mismo microcontrolador podemos utilizar su IDE gratuito para poder programar el robot en Arduino.

En esta parte tenemos dos elementos importantes.

- ***Microcontrolador ATmega168V AVR 32K FLASH 2K SRAM***

El elemento central del robot, será el encargado de almacenar y cargar la información que queramos introducirle.

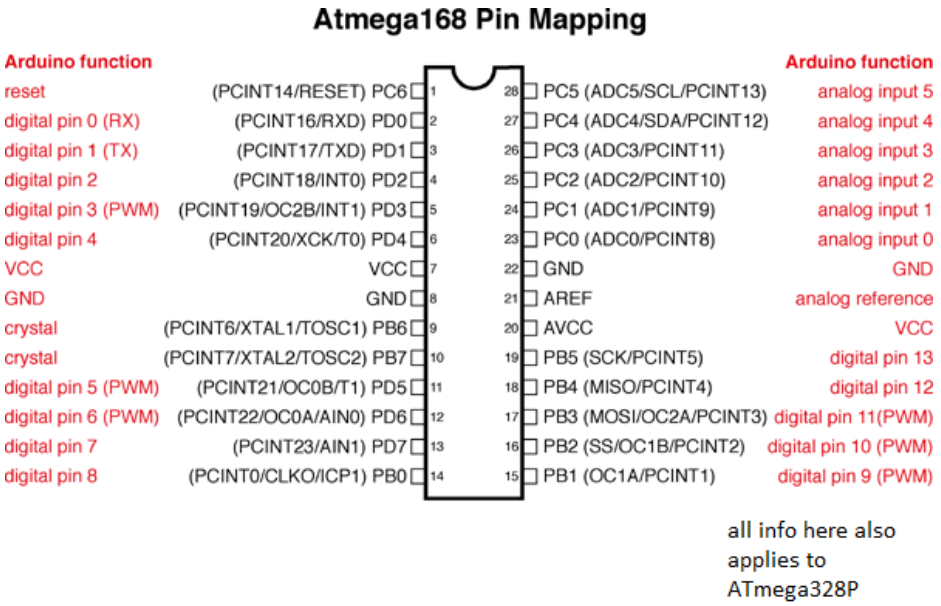


Figura 4: Mapeado de pines entre Arduino y ATmega168V

En la imagen anterior *(Figura 4)* podemos observar la correspondencia de pines de Arduino con cada una de las patillas del microcontrolador ATmega168V.

Algunas de las características del microcontrolador son:

• Dimensiones	37.4 x 6.67 x 3.28 mm
• Número de pines	28
• Frecuencia máxima	20 MHz
• Tamaño de la memoria del programa	1 kB, 32 kB
• Tamaño RAM	2 kB
• Rango de temperaturas	-40 °C y +85 ° C
• Tensión de alimentación típica	1.8 ⟶ 5.5 V
• Tipo de montaje	Orificio pasante

❖ ***Multiplexor/Demultiplexor Analógico CD4067BE-LOGIC, 16-CH***

Este componente será el que se encargue de controlar las entradas y salidas de los distintos sensores de los que consta el robot.

Las características más importantes son:

• Número de pines	24
• Número de canales	16
• Corriente de alimentación	100 µA
• Potencia de disipación	500 mW
• Tiempo de apagado	130 ns
• Tiempo de encendido	190 ns

• Rango de temperaturas	-55 °C y +125 ° C
• Tensión de alimentación típica	3 → 18 V
• Tipo de montaje	Orificio pasante

En la siguiente imagen *(Figura 5)* podemos observar el esquemático de este elemento junto a la asignación de los pines.

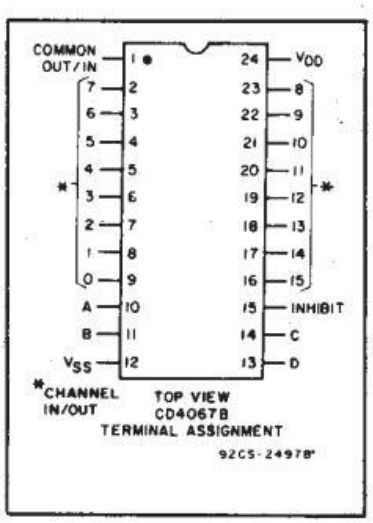


Figura 5: Multiplexor/Demultiplexor Analógico CD4067BE-LOGIC

3.2.1.2. Gestor de arranque

Este apartado corresponde a la conexión entre el robot (placa) y el ordenador. Existen varias opciones que nos permiten comunicarlos para que de este modo podamos escribir las instrucciones y así poder programar el robot o realizar pruebas de funcionamiento con él.

Para elegir cual es el mejor hardware de programación que mejor se ajusta a nuestras necesidades veremos las ventajas e inconvenientes de algunas posibles soluciones.

3.2.1.2.1. Gestor de arranque normal y FTDI/XBee

Esta opción es la misma que la configuración para SkyMega. Es necesario un cable FTDI que permita programar el chip.

SkyMega es un proyecto de la Asociación de Robótica de la UC3M que consiste en una pequeña tarjeta microcontroladora cuya aplicación más destacada es la programación de robots modulares.

El cable consta de un puerto USB en uno de los extremos y de un conector de 6 pines en el otro extremo.

- Ventajas
  - Robot de bajo coste. Este aspecto se refleja en la robótica de enjambre; si ya se posee el cable podemos usar el mismo cable para todos los robots.
  - Permite la programación en el aire (OTA).
  - No SMD.
  - Robusto y probado.
- Inconvenientes
  - Precio del cable FTDI.

SMD (Surface Mounting Device) es un tipo de soldadura mediante el cual los componentes se unen directamente a la placa sin necesidad de realizar agujeros (Orificio pasante). Aunque posee ciertas ventajas con respecto al otro tipo de unión como son su reducido tamaño y la ausencia de hilos; en nuestro caso es preferible usar orificio pasante ya que debido a su futuro carácter comercial es posible que todos aquellos que quieran construirse su propio robot no sepan o no tengan los medios necesarios para utilizar la soldadura SMD.

3.2.1.2.2. Gestor de arranque v-USB

Esta posibilidad es muy similar a USnooBie. Consiste en un kit microcontrolador que no requiere ningún tipo de programador AVR o convertidores de USB a serial para cargar y ejecutar código compilado. Su diseño de hardware permite al usuario desarrollar dispositivos USB de bajo costo con microcontroladores ATmega AVR de Atmel.

El V-USB *(Figura 6)* es una implementación de un dispositivo USB de baja velocidad para microcontroladores AVR de Atmel, por lo que es posible la construcción de hardware USB con casi cualquier microcontrolador AVR, no requiriendo ningún chip adicional.

- Ventajas
  - Coste 0 €
- Inconvenientes
  - Utiliza 2 pines: PORTD2 (D2) y PORTD7 (D7).
  - No permite la programación en el aire.
  - No ha sido probado en el diseño; por lo que no estamos seguros de que realmente funcione para nuestro caso.
  - No se sabe si es 100 % transparente para la IDE de Arduino. En principio parece que solo hay que cambiar una preferencia del archivo1.
  - No se sabe si puede introducir nuevas posibilidades de fallo (Es decir, incompatibilidades al funcionar el código).

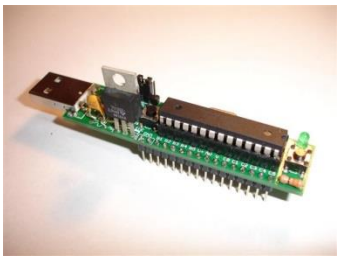


Figura 6: v-USB

3.2.1.2.3. Gestor de arranque normal y USB/XBee

Esta opción consiste en integrar a la PCB un chip FTDI. Un chip FTDI es un circuito integrado (creado por la empresa del mismo nombre) que se usa comúnmente en los dispositivos electrónicos que utilizan los microcontroladores.

Permiten la migración USB mediante la combinación USB-Serial (USB-RS232) y las soluciones USB-FIFO con drivers USB libres.

- Ventajas
  - Tiene un precio más barato que el cable (6 € de chip frente a 16 € del cable FTDI).
  - Permite la programación en el aire.
- Inconvenientes
  - Soldadura SMD. En nuestro caso es un inconveniente a la hora de incorporarlo; puesto que nos dificulta el soldado a la placa.

3.2.1.2.4. Gestor de arranque Foca

En este caso nos encontramos con un cable que consta de un puerto USB en uno de los extremos y de un puerto USB mini en el otro. El puerto mini USB se conecta al adaptador Foca *(Figura 7)* que además de tener un conector de 5 pines en el otro extremo permite colocar un XBee con lo que facilitamos la programación.

Figura 7: Conector Foca

Estas son las distintas posibilidades que tenemos para elegir la forma en la que comunicar el ordenador al robot.

De las cuatro posibilidades, el gestor de arranque v-USB no ha sido probado. Habría que realizar un test para ver si funciona sin problemas. Tendríamos que considerar si necesitamos programación OTA; de ser así se podría considerar como opción. Esto se puede investigar para una futura revisión.

El chip FTDI sería una buena opción de no ser por el problema de la soldadura, que dificulta su montaje.

El robot va a usar unos módulos de transmisión inalámbrica XBee (de los que hablaremos más adelante). Por lo que la mejor opción sería venderlo con un cable USB mini y el adaptador Foca. Esto nos permitiría usarlo tanto para programar el robot como para conectar el módulo XBee al ordenador. Nos decantamos finalmente por esta opción.

3.2.2. Gestión de energía

En este apartado veremos los elementos correspondientes a la alimentación del robot y cómo gestionar esta energía.

3.2.2.1. Requerimientos

En primer lugar deberemos establecer un presupuesto aproximado de la alimentación que necesitaremos en función de los distintos componentes que llevará incorporado el robot.

Presupuesto de energía

Dispositivo	Nº	Corriente media (mA)	Corriente media total (mA)
ATmega168	1	10	10
SHARP GP2D12 (Sensor IR)	5 (Var)	40	200
HC-SR04 (Sensor ultrasónico)	3 (Var)	30	90
Motor DC	2	50	100
XBee	1	50	50
Led IR para odometría	4	20	80
Led IR para tierra firme (LD271)	3	130	390
Led RGB para límite de cámara	3	50	150
		Max Total	**1070 mA**

El robot fabricado no lleva todos los sensores de los cuales hacemos la estimación; sin embargo hemos dicho anteriormente que este robot pretende ser muy versátil. En función de la situación requerida podemos necesitar más o menos elementos.

Dispondremos a su vez de un panel que habilite los distintos dispositivos. Estos serán:

- Sensor IR
- Leds IR
- Encoders
- Radio

❖ ***Interruptor de 4 vías, dil, slice 78B04T***

Este interruptor cuádruple *(Figura 8)* será el que nos permita conmutar las distintas partes de las que consta el robot (sensores, encoders, leds y radio). Características:

• Tipo de interruptor	DIP
• Número de interruptores	4
• Corriente nominal	4 A
• Montaje	Orificio pasante
• Rango de temperaturas	-40 °C y +85 ° C

Figura 8: Interruptor cuádruple 78B04T

3.2.2.2. Batería

Tenemos que tener en cuenta varios aspectos con respecto a la batería. Para alimentar el robot usaremos una batería LiPo *(Figura 9)*. Este tipo de baterías son utilizadas normalmente en coches de radiocontrol.

❖ ***Batería LiPo Pack Turnigy 1300mAh 2S 20C***

Ésta será la batería que llevará nuestro robot. Sus características son:

• Capacidad mínima	1300 mAh
• Configuración	2S1P / 7.4v / 2Pilas
• Constante de descarga	20C
• Pico de descarga (10sec)	30C
• Peso del pack	82 g

• Dimensiones	70 x 35 x 15 mm
• Plug de carga	JST-XH
• Plug de descarga	XT60

Figura 9: Batería LiPo pack Turnigy

Para evitar comprar una nueva batería cada vez que se nos gaste, será necesario recargarla una vez se consuma. Es por eso que utilizaremos un cargador por USB *(Figura 10)* para poder reutilizar la batería.

❖ ***Equilibrador y cargador Turnigy 2S-3S***

Figura 10: Cargador de batería Turnigy

Las características de nuestro cargador son:

• Entrada	12∼15 V DC
• Corriente de carga	800 mA

3.2.2.3. Protecciones

Otro factor importante es la protección y distribución de la energía. Este bloque incluye la monitorización, protección y convertidor DC-DC. Este apartado consta de tres partes y disponemos de varias posibilidades a la hora de elegir una opción u otra.

- Convertidor DC
- Detector de baja tensión
- Fusible

3.2.2.3.1. Convertidor DC

A la hora de gestionar la energía tenemos dos opciones posibles:

- Utilizar un regulador de voltaje lineal.
- Usar una fuente conmutada.

Veamos cada una de ellas de forma general y apliquémoslo a nuestro caso con los componentes elegidos.

Fuente conmutada

- ***Convertidor DC-DC 2.25 A PTH08000WAZT***

Aquí nos encontramos con el convertidor DC *(Figura 11)*. Algunas de sus características son:

• Dimensiones	18.92 x 12.57 x 8.25mm
• Número de pines	6
• Número de salidas	1
• Función del regulador	Reductor
• Regulador de conmutación	Sí
• Rango de temperaturas	-40 °C y +85 ° C
• Tipo de encapsulado	Módulo DIP
• Tipo de montaje	Montaje en superficie

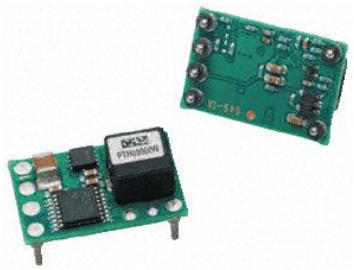


Figura 11: Convertidor DC-DC PTH08000WAZT

Debemos tener en cuenta de este convertidor que:

- Regulación de salida de tensión de 0.9 V a 5.5 V, programable a través de resistor externo.
- Control remoto de Encendido/Apagado.
- Protección ante sobrecorriente y sobretemperatura.
- Bloqueo de tensión mínima (UVLO).
- I (espera) < 1 mA. (La corriente que el convertidor dibuja la podemos poner en modo apagado).

Tras ver un poco sus características pasamos a ver las ventajas e inconvenientes de usar este componente.

- Ventajas
  - Eficiencia *(Figura 12)*: Aproximadamente entre el 80% y el 90% para PTH08000WAZT (V_{IN}= 12 V, V_{OUT}= 5 V).

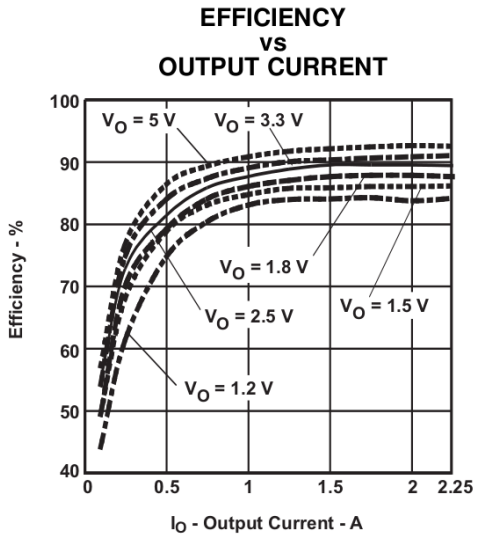


Figura 12: Curva convertidor Eficiencia-Corriente de salida

  - Pin powerOff.
- Inconvenientes
  - Alto coste (10 €).
  - Menos estándar.
  - Es ruidoso. (14mV rms)

Parece que no hay necesidad de utilizar un mosfet para conducir el pin on/off de PTH08000W: el mosfet que se recomienda en la hoja de datos tiene una resistencia de encendido de varios Ω, lo que significa que la salida de un chip lógico sería suficiente.

Regulador de voltaje lineal

- ***Regulador lineal MCP1700-3302E/TO 3.3V***

Aquí nos encontramos con la otra opción. Veamos primero sus características:

• Número de pines	3
• Número de salidas	1
• Corriente de salida máxima	0.25 A
• Corriente de salida	250 mA
• Tensión de entrada máxima/mínima	6 V / 2.3 V
• Tensión de salida	3 V
• Rango de temperaturas	-40 °C y +125 ° C

Es el momento de considerar las ventajas e inconvenientes de usar este componente.

- Ventajas
  - Bajo coste.
  - Simple.
  - Sin ruidos.
  - Disponible.
- Inconvenientes
  - La eficiencia puede ser muy baja debido a altas diferencias de tensión.
  - Cuando pasamos la nominal de 7.4 V a la batería de 5 V, la eficiencia es 5/7.4 = 67%.
  - Problema: cuando usamos fuente de alimentación externa (podría ser tan alta como unos 12 V) la eficiencia podría ser de 5/12 = 42%.
  - Problema: no hay pin powerOff para desconectar el convertidor de la LiPo cuando está bajo de tensión.

Finalmente y estudiadas ambas opciones hemos decidido utilizar la fuente conmutada; en el robot usamos el convertidor DC-DC 2.25 A PTH08000WAZT que es un modelo estándar (RS), tiene un coste relativamente bajo y una alta eficiencia. Será por tanto la opción elegida.

3.2.2.3.2. Detector de baja tensión

Las opciones para solucionar este apartado son:

- Circuito $V_{Ref\ Zener}$ + Comparador.
- Detector de tensión.

$V_{Ref\ Zener}$ + Comparador

Aquí tenemos la opción de utilizar un circuito con el objetivo de proteger a la batería de sobredescargas. También evitaría daños en la batería o explosión en un cortocircuito.

El esquema de este circuito, así como distintas medidas tomadas podemos verlas con más detalle en el apartado de "Pruebas y funcionamiento". Los componentes más importantes de este circuito los mostramos a continuación.

- ***VRef ajustable 2.495V a 36V 100mA TL431***

Las características más importantes del regulador de tensión ajustable son:

• Número de pines	3
• Tipo de referencia	Ajustable

• Montaje	Orificio pasante
• Corriente de salida máxima	100 mA
• Tensión de entrada máxima	37 V
• Tensión de salida máxima/mínima	36 V / 2.5 V
• Rango de temperaturas de funcionamiento	0 °C y 70 ° C

❖ ***Comparador IC único LM311P***

El comparador *(Figura 13)* tiene las siguientes especificaciones:

• Número de pines	8
• Tipo de comparador	Alta velocidad
• Rango de tensión	3.5 V y 30 V
• Rango de temperaturas de funcionamiento	0 °C y 70 ° C

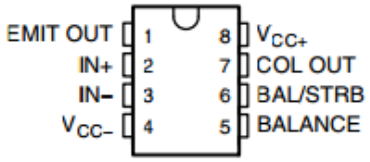


Figura 13: Comparador IC LM311P

Si el offset de equilibrio no se utiliza, los pines BAL/STRB y BALANCE deben ser cortocircuitados.

Finalmente, y tras estudiar esta opción, hemos decidido descartar este circuito puesto que existen otras soluciones más fáciles que indicaremos en el siguiente punto.

Detector de baja tensión de los circuitos integrados (ICs)

Existen diversas soluciones de chip únicas para evitar usar la $V_{Ref\ Zener}$ + Comparador:

- http://docs-europe.electrocomponents.com/webdocs/0c30/0900766b80c3081a.pdf
- http://es.rs-online.com/web/c/?sra=oss&searchTerm=low+power+detector&x=0&y=0
- http://search.digikey.com/scripts/dksearch/dksus.dll?vendor=0&keywords=low+power+detector

Finalmente se ha elegido utilizar el detector de tensión TC54 *(Figura 14)* para este cometido.

❖ ***Detector de tensión 2.7V TC54VN2702EZB***

Este componente será el encargado de controlar la tensión de nuestra batería en todo momento; nos indica en qué estado se encuentra y su capacidad. Características:

• Número de pines	3
• Dimensiones	4.71 x 3.62 x 4.62mm

• Tensión de alimentación de funcionamiento máxima/mínima	10 V / 0.7 V
• Tensión máxima/mínima controlada	5.5 V / 1 V
• Señal de subtensión	2.7 V con histéresis
• Rango de temperaturas	-40 °C y +85 ° C
• Tipo de montaje	Orificio pasante

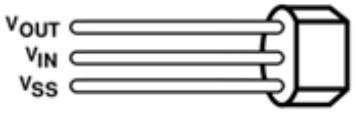


Figura 14: Detector de tensión TC54VN2702EZB

3.2.2.3.3. Fusible

❖ ***Fusible PTC reajustable 1.1 A radial RKEF110***

El fusible *(Figura 15)* es el encargado de proteger al circuito ante las posibles corrientes excesivas que puedan producirse por la acción de la batería. Características:

• Corriente de disparo	2.2 A
• Corriente de mantenimiento	1.1 A
• Corriente máxima	40 A
• Potencia de disipación	2.2 W
• Resistencia máxima/mínima	0.47 Ω / 0.17 Ω
• Tensión nominal	60 V
• Tiempo de retardo de disparo	3 s

Figura 15: Fusible PTC reajustable RKEF110

Finalmente y tras elegir los distintos componentes que forman parte de las protecciones y el convertidor, nos disponemos a montar el circuito. Con el conseguimos:

- Convertidor DC/DC con protección de sobrecorriente y sobretemperatura.
- Bloqueo de tensión mínima (UVLO) a 6.75 V (3.375 V/pila) con indicador Led.
- Seguridad polyfuse (PPTC).

El esquema del circuito podemos observarlo en la siguiente imagen *(Figura 16).*

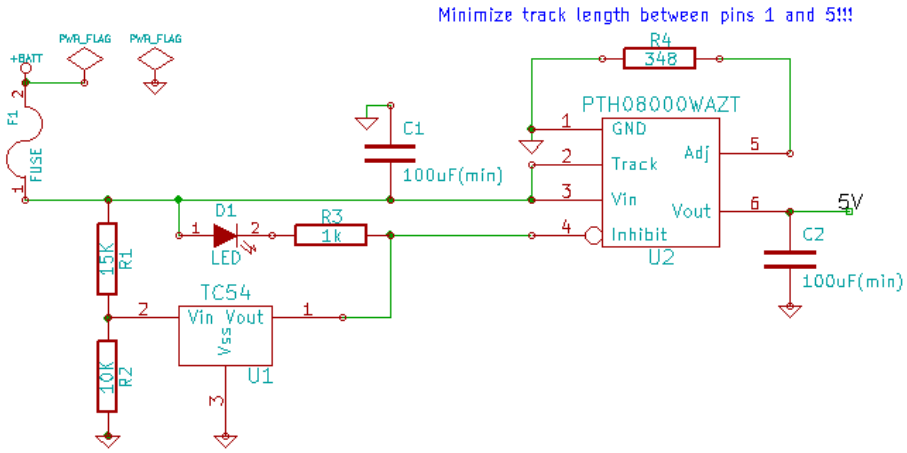


Figura 16: Circuito convertidor DC-DC y protecciones

El TC54 detecta la tensión de la batería y apagan el convertidor DC-DC cuando la batería está casi vacía (6.75V).

El PPTC se utiliza como una protección extrema (solamente si el cortocircuito ocurre antes del convertidor DC-DC). Cualquier cortocircuito en la salida del convertidor se gestionará por sí mismo. Esto es así porque la batería de nuestro robot está diseñada para coches de radiocontrol, con lo que la corriente es demasiado grande para que pueda soportarlo. Por esta razón es necesario establecer esta protección.

3.2.3. Comunicaciones

Esta parte corresponde con la comunicación entre el robot y el ordenador; el cual le dará las órdenes necesarias y permitirá la transmisión de datos entre ambos.

El elemento principal de este apartado son los módulos XBee *(Figura 17)* que serán los que permitan la comunicación de forma inalámbrica entre el robot y el ordenador.

- ***Módulo XBee, antena de alambre, XB24-AWI-001, 1 MW***

Las características de nuestros módulos son:

• Rango de temperaturas	-40 °C y 85 ° C
• Tipo	Módulo XBee RF

Figura 17: Módulo XBee XB24-AWI-001

Este kit destaca por:

- Integridad de los datos a larga distancia.
- Bajo consumo de energía.
- Creación de redes y seguridad avanzada.
- ADC y soporte de línea I/O.

Tendremos dos módulos distintos: uno como receptor y otro como transmisor. Uno de ellos se conecta al ordenador mediante el cable USB y el adaptador Foca; el otro se sitúa en el robot. Ambos módulos son iguales y sirven como emisor-receptor (del mismo modo que un walkie talkie).

Cuando se programa el robot es necesario utilizar el adaptador Foca para conectarlo al robot y así introducirle el programa en el que se haya trabajado (es importante destacar que cuando se introduce el programa en el chip AtMega los XBee deben de estar quitados).

Una vez el programa se encuentra en el robot, uno de los XBee es el que se encuentra en el robot y el otro en el ordenador, de tal forma que el del ordenador manda la información a su XBee y éste al XBee del robot. La comunicación entre ambos es bidireccional, permitiendo a ambos funcionar a la vez como transmisor y receptor.

Veamos pues el esquema de cada uno de ellos.

En primer lugar nos centramos en el transmisor XBee *(Figura 18)*. Éste será el que conecte al ordenador mediante el cable USB mini y el adaptador Foca indicado anteriormente.

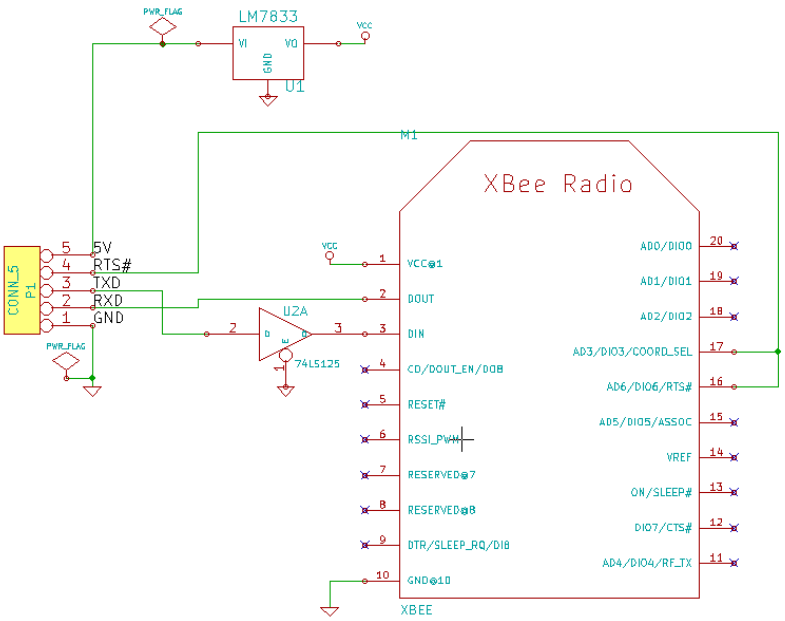


Figura 18: Transmisor XBee

El siguiente que vemos corresponde con el receptor XBee *(Figura 19)*. Éste será el que se coloque en el robot.

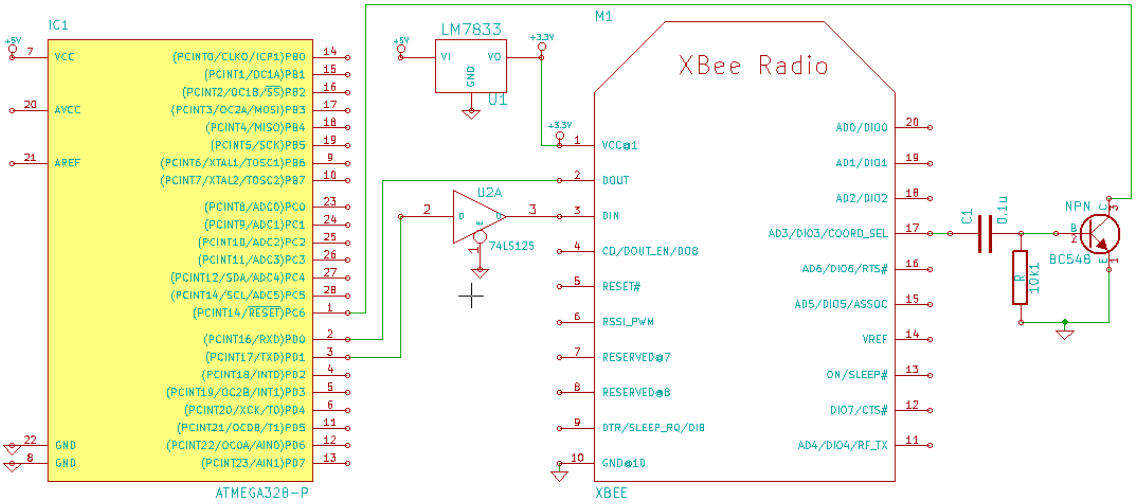


Figura 19: Receptor XBee

Tras ver ambos módulos, lo siguiente que deberemos hacer será configurarlos para que puedan comunicarse entre sí. Esto podemos verlo en el apartado de "Pruebas y funcionamiento".

Otro elemento que se planteó durante el desarrollo del proyecto fue la incorporación de un módulo de Bluetooth *(Figura 20).*

- ***Módulo de puerto serie Bluetooth para Arduino JY-MCU***

Este componente permitiría al robot poder comunicarse mediante Bluetooth. Aunque disponemos del componente, para este primer prototipo no se ha utilizado; sin embargo sí que se tiene en consideración para futuras ampliaciones.

Figura 20: Módulo Bluetooth JY-MCU

3.2.4. Sensores

Este apartado corresponde a los distintos sensores que lleva equipado el robot. Podemos distinguir tres tipos distintos de sensores:

- Sensor infrarrojo.
- Sensor ultrasonidos.
- Encoders o contadores.

Nuestro robot puede está diseñado para llevar más sensores de los que lleva en un principio. Para este primer prototipo los sensores de los que disponemos son 1 infrarrojo, 2 ultrasonidos y 4 encoders.

❖ ***Módulo ultrasónico sin zona ciega SDM-IO***

El robot llevará en un principio dos sensores de ultrasonidos *(Figura 21).* Irán colocados en la parte delantera, a ambos lados para tener un mayor radio de visión. Sus características son:

• Rango de alcance	0 cm y 150 cm
• Tensión de alimentación máxima/mínima	5.5 V / 3.8 V
• Consumo de corriente	8 mA
• Frecuencia ultrasónica	40 kHz
• Disparo de ancho de pulso	10 µs
• Ángulo de sensor	15 grados

Figura 21: Sensor ultrasonidos

❖ ***Sensor de distancia analógico GP2Y0A21YK0F (de 10cm a 80cm)***

Este sensor es de tipo infrarrojo *(Figura 22)* y se colocará en la parte delantera del robot entre los sensores ultrasónicos. Características:

• Rango de alcance	10 cm y 80 cm
• Estilo de sensor	Bloque
• Fuente de la luz	Led Infrarrojo
• Tensión de alimentación máxima	5.5 V
• Rango de temperaturas	-10 °C y 60 ° C
• Tipo de salida	Analógico

Figura 22: Sensor infrarrojo

- ***Fototransistor PT480F***

Aquí nos encontramos con los encoders *(Figura 23)* que se sitúan en las ruedas y se encargaran de controlar la velocidad a la que se mueve el robot en función del paso de la luz por las cavidades de las mismas. Serán necesarios cuatro; es decir, dos por rueda. Como características destacamos:

• Número de pines	2
• Número de canales	1
• Corriente de luz máxima	3000 µA
• Corriente de oscuridad máxima	100 nA
• Mínima longitud de onda detectada	860 nm
• Montaje	Orificio pasante
• Tiempo de subida típico	3000 ns
• Tiempo de bajada típico	3500 ns
• Polaridad	NPN

Figura 23: Fototransistor

- ***Led IR de Vista lateral de emisión, GL4800E0000F***

Estos leds *(Figura 24)* se colocan junto a los fototransistores para que éstos puedan captar su luz. Hay uno por cada fototransistor. Tiene como características:

• Número de pines	2
• Montaje	Orificio pasante
• Longitud de onda de pico	950 nm

Figura 24: Led IR

El voltaje directo de estos leds a 20 mA es de 1.2 V y a 17.4 mA es de 1.19 V.

3.2.5. Actuadores

En esta parte indicaremos aquellos componentes que se encargan de mover el robot o algún sensor; así como elementos relacionados con los mismos.

- ***Micro motor-reductor de metal 250:1***

Tendremos un total de dos motores *(Figura 25)* y serán los encargados de hacer mover al robot. Llevarán acoplados unas ruedas impresas con las que proporcionaremos el movimiento. Sus características son para una tensión de 6 V:

• Corriente de parada	1.6 A
• Velocidad sin carga	120 rpm
• Par de arranque	4.3 kg/cm
• Tipo de motor	Alta potencia (HP)

Figura 25: Motor-reductor

- ***Emisor infrarrojo, 950 nm, 5mm LD271***

Son tres leds *(Figura 26)* que se pusieron con el objetivo de que fuesen captados por el mando (wiimote) de la consola Wii de Nintendo. De este modo podríamos saber la posición del robot en cada momento colocando el mando en el techo y captando la luz de estos emisores. Tiene como características:

• Número de pines	2
• Longitud de onda de pico	950 nm
• Material del led	GaAs
• Tipo de montaje	Orificio pasante

Figura 26: Led emisor de posición

❖ ***Driver de motor de cuatro canales L293DNE***

Se encarga de controlar y gestionar el funcionamiento de los motores. Características:

• Número de pines	16
• Rango de temperaturas	0 °C y 70 ° C
• Tensión de alimentación máxima/mínima	36 V / 4.5 V
• Tipo de montaje	Orificio pasante

❖ ***Mini Servo TowerPro SG90 9G con accesorios***

El servo *(Figura 27)* es el componente encargado de hacer girar de posición al sensor infrarrojo que se situará encima. Tiene un funcionamiento similar a los motores en cuanto a giro. Sus características son:

• Dimensiones	3.2 cm x 3 cm x 1.2 cm
• Par máximo (4.8 V)	1.2 kg/cm
• Rango de temperaturas	-30 °C y 60 ° C
• Tensión de funcionamiento	3.5 V / 7.2 V

Figura 27: Mini servo y accesorios

El mini servo viene incluido con unas hélices de plástico que sirven a modo de adaptadores. Éstas nos servirán para colocar el sensor de infrarrojos encima.

3.3. Software utilizado

Todos los circuitos necesarios para realizar el robot han sido creados mediante el programa de software gratuito de KiCad que se encuentra disponible para los sistemas operativos de Windows y Linux.

Este programa es un conjunto de aplicaciones que permite un entorno fácil y flexible con el cual podemos crear y modificar una gran cantidad de esquemas de conexionado y circuitos impresos.

KiCad permite a su vez diseñar estos circuitos impresos con múltiples capas y visualizarlos en 3D.

KiCad *(Figura 28)* está dividido en cinco aplicaciones distintas, cada una de ellas encargada de una función concreta:

- EeSchema: Editor de esquemas.
- Cvpcb: Asociación entre los componentes del esquema y los módulos correspondientes del circuito impreso.
- Pcbnew: Editor de circuitos impresos (PCB).
- GerbView: Visualizador de archivos Gerber.
- Bitmap2Component: Crea un componente (para EeSchema) o una huella (para Pcbnew) que muestra un dibujo BB&W

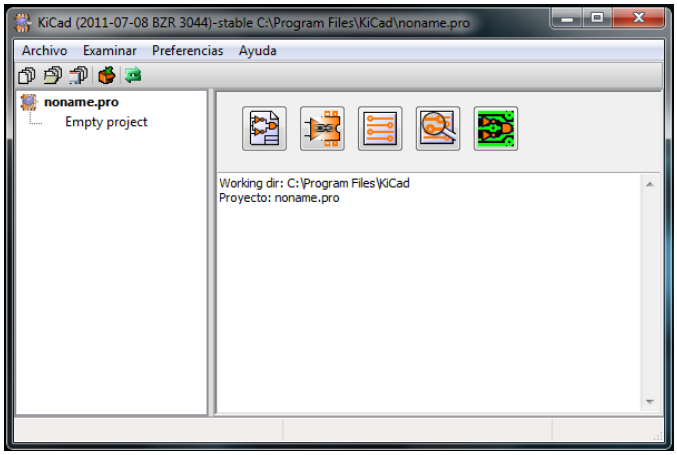


Figura 28: Entorno KiCad

En la parte izquierda se sitúa la ventana del árbol del proyecto; en ella nos muestra el proyecto activo en ese momento y todos los archivos que contiene pertenecientes a los distintos programas.

En la parte derecha abajo tenemos una ventana de mensajes y encima de ella se encuentran los botones de acceso a las aplicaciones.

Para poder trabajar con KiCad será necesario crear un nuevo proyecto que permita gestionar cada una de estas aplicaciones mediante parámetros comunes (como las

bibliotecas utilizadas en el esquema y en los circuitos impresos entre otros). La extensión de este fichero será del tipo .pro.

A continuación explicaremos cada una de las partes de forma más detallada.

3.3.1. EeSchema

Accedemos a él mediante el botón de acceso de KiCad. De este modo se nos abre una nueva ventana correspondiente a EeSchema *(Figura 29)*.

A la hora de crear nuestra placa, este programa será el que usaremos en primer lugar. Se encarga de diseñar y editar los esquemas de circuitos electrónicos.

Además de tener su propia biblioteca, podemos editar y crear distintos componentes; así como importar y exportar componentes en las bibliotecas.

EsSchema se asocia a su vez con Pcbnew proporcionándole el fichero *.net* (Netlist) que describe el esquema del circuito impreso a realizar. Este tipo de ficheros se encargan de dar la lista de componentes y la lista de conexiones resultantes en un esquema que hemos creado previamente. Este fichero es muy importante puesto que más adelante será usado por otros programas.

Para diseñar totalmente la placa es necesario pasar por cada uno de estos programas. Cada uno aporta algo nuevo al siguiente, lo que nos permitirá completarlo.

Cuando abrimos el programa nos encontramos con lo siguiente:

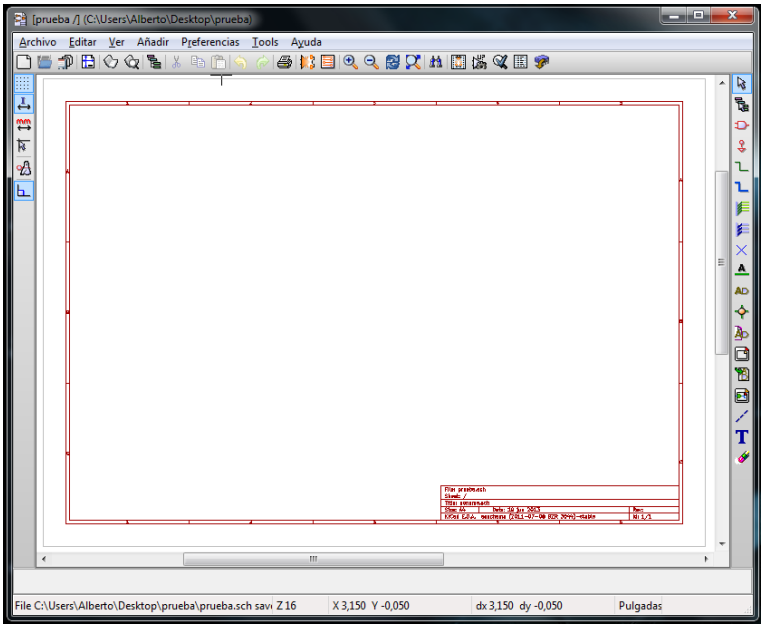


Figura 29: Entorno EeSchema

Como se puede observar el programa cuenta con una gran zona central que será la zona de trabajo en la cual iremos construyendo nuestro esquema.

En el margen superior y en ambos lados de la zona central nos encontramos con distintas barras de herramientas que nos facilitaran el trabajo para la construcción del esquemático.

- Barra de herramientas superior: Aquí se encuentran la barra de tareas correspondiente a las funciones principales. Podemos crear, abrir y guardar el esquema, copiar o suprimir elementos, zoom, crear archivo Netlist o abrir Cvpcb y Pcbnew; entre otros.
- Barra de herramientas derecho: En este lado está enfocado a colocar/borrar componentes, cables, buses, etiquetas de red, textos… En el caso de utilizar más de un esquema (esquema multihoja) está barra nos permite navegar entre las distintas hojas, así como crear otras nuevas en la jerarquía.
- Barra de herramientas izquierdo: Esta barra se encarga de los botones de visualización de la rejilla, cursor, unidades, pins “invisibles” y de las direcciones de cables y buses.

Es importante destacar también que utilizando el botón derecho del ratón podemos acceder a un menú que variará en función de lo que el puntero esté señalando en ese momento (etiqueta, componente o nada). Con esta opción podemos por ejemplo editar una etiqueta o componente o bien cambiar su orientación.

El programa posee su propia biblioteca con una gran cantidad de elementos disponibles; sin embargo, EsSchema tiene un editor con el cual podremos crear y editar nuestros propios componentes si no encontramos el que necesitamos.

Tras crear un archivo (extensión .sch) y haberlo guardado podemos comenzar a construir nuestro esquema mediante las barras de herramientas que hemos mencionado anteriormente.

En función de la complejidad de nuestro esquema, ésta se representará en una o varias hojas. Lo más común es que ocupen más de una hoja, desarrollándose así un sistema jerárquico (con un esquema raíz o principal y un conjunto de sub-esquemas asociados a este). Cada hoja representa un fichero propio y el conjunto de todas ellas constituye un proyecto para EeSchema.

Este programa tiene muchas más funciones. Gracias a EsSchema podemos además:

- Detectar errores, incoherencias u olvidos en el esquema. Esto lo conseguimos mediante el control E.R.C
- Generar automáticamente la lista de componentes utilizada.
- Crear archivos Netlist para simular el funcionamiento del esquema en otros programas (ej. Pspice).
- Crear archivos Netlist para realizar circuitos impresos (Pcbnew).

En la siguiente imagen podemos apreciar el desarrollo que tiene lugar en la creación de un esquema *(Figura 30)* y su posterior uso.

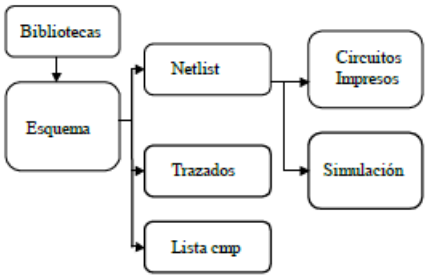


Figura 30: Cadena de desarrollo

Todo el proceso comienza a partir de las bibliotecas de componentes; de ahí creamos el esquema que más tarde se utilizará para producir el archivo Netlist y que usaremos finalmente en Pcbnew para crear nuestro circuito impreso.

Una vez que hemos hecho nuestro esquema y se ha comprobado que no contiene errores, lo siguiente que deberemos hacer será la creación del archivo Netlist.

3.3.2. Cvpcb

Cvpcb *(Figura 31)* es la segunda aplicación disponible de KiCad. Este programa utiliza el fichero Netlist anterior y permite completarlo de tal forma que asocie a cada componente del Netlist un módulo que lo represente en la placa de circuito impreso.

Asocia de manera interactiva módulos y componentes mediante ficheros de equivalencia. El resultado es la obtención de otro fichero Netlist completo (con referencia a los módulos) y de un fichero auxiliar para la asociación de componentes con extensión .cmp.

Cuando abrimos el programa nos encontramos con el siguiente entorno:

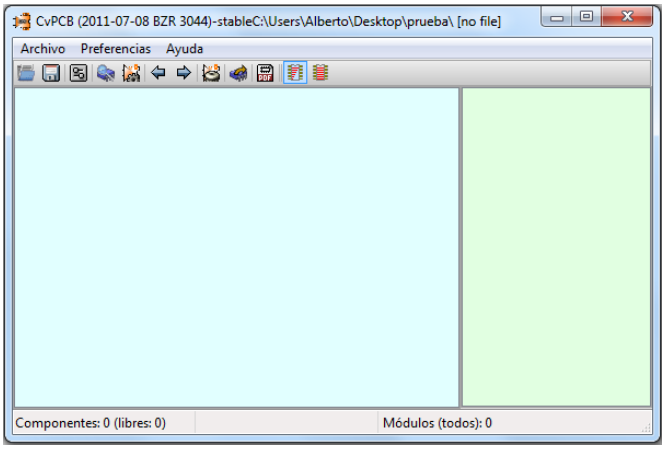


Figura 31: Entorno Cvpcb

Nos encontramos en la parte superior del programa con una barra de herramientas en la que encontramos algunas de sus funciones principales como lo son seleccionar el fichero Netlist, crear los ficheros *.cmp* y *.net* modificados y completos, asociar, mostrar y borrar componentes; entre otros.

También nos encontramos a la izquierda con una ventana correspondiente con los componentes. Cuando tenemos un archivo cargado aquí se nos muestra la lista de componentes pertenecientes a la Netlist.

La ventana de la derecha corresponde con los módulos. En ella aparece la lista de módulos pertenecientes a las bibliotecas leídas. Es necesario cargar una biblioteca para poder realizar la asociación.

3.3.3. Pcbnew

El tercer programa que nos encontramos en KiCad corresponde a Pcbnew *(Figura 32)*. Esta aplicación nos permite crear circuitos impresos a partir de un archivo Netlist que indicará el esquema de diseño del circuito que queremos realizar.

Con Pcbnew podemos gestionar las bibliotecas de módulos que serán cargados automáticamente al leer los archivos Netlist.

Del mismo modo permite modificar el esquema de forma integral: modificar módulos (tanto antiguos como nuevos), añadir nuevos componentes o eliminar pistas erróneas; entre otros. Podemos crear un circuito de manera sencilla e intuitiva; señalando automáticamente los posibles errores que cometamos en el diseño de las pistas.

Otra característica de Pcbnew es que sitúa todos los elementos que puedan componer nuestro circuito respetando las formas y mostrándolos de distintas formas: mostrar el contorno, trazos continuos, márgenes de aislamiento eléctricos…

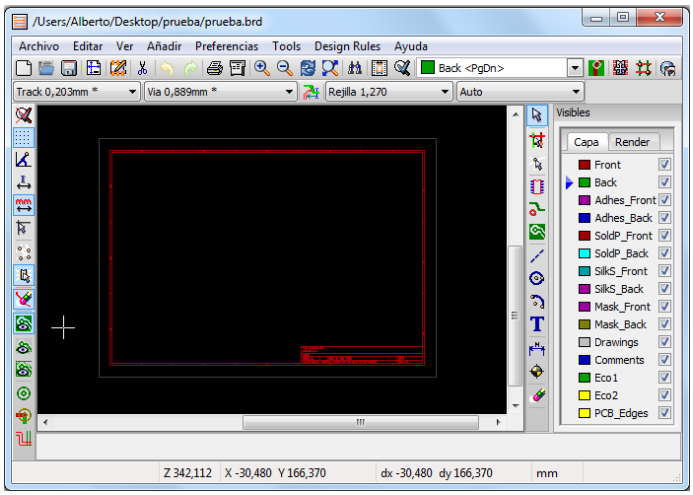


Figura 32: Entorno Pcbnew

Lo primero que podemos apreciar es que el programa se divide en distintas barras de herramientas, distinguiéndose un total de cuatro distintas:

- En la zona superior del programa se sitúa la barra de menús. Aquí podemos abrir y/o guardar esquemas, elegir de las bibliotecas de trabajo, ajustar el tamaños, creación de ficheros de taladro e incluso la visualización en 3D de nuestro circuito *(Figura 33)*.

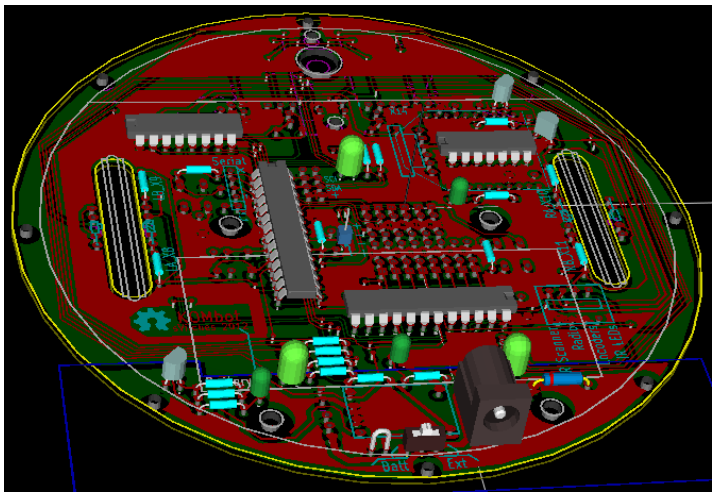


Figura 33: Circuito en 3D

- Debajo de la barra de menús se encuentra la barra de tareas. En esta zona se sitúan los iconos correspondientes a las funciones principales de Pcbnew.
- En la zona de la derecha del programa podemos ver otra barra de herramientas.
- En la parte izquierda se encuentra la última barra. Ésta se encarga de algunas de las opciones de control y visualización.

En el centro del programa se encuentra la zona de trabajo en la que realizaremos nuestro circuito usando las herramientas descritas anteriormente.

Una vez tenemos creado nuestro circuito impreso, se guardará el archivo en formato *.brd*. Para que el proceso esté completo será necesario la creación de ficheros de taladrado y de fototrazado Gerber.

Un fototrazador *(Figura 34)* está compuesto por una mesa X-Y que es controlada por unos servomotores de precisión y sobre el cual se sitúa una película de alto contraste. Se dirige una fuente de luz a través de una rueda con diferentes aberturas y que atraviesa un obturador y de ahí a la película. Los movimientos de la mesa vienen dados por los comandos dados por Gerber en forma de controlador. De este modo se traza el dibujo correspondiente.

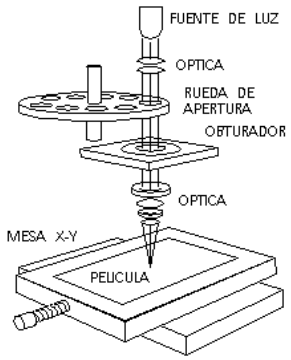


Figura 34: Fototrazador

Resumiendo, para la creación de un circuito impreso mediante este conjunto de aplicaciones; los pasos a seguir son:

1. Creación del esquema del circuito el cual queremos realizar su versión impresa. Una vez que tenemos hecho nuestro circuito y se comprueba que es plenamente operativo, podemos empezar a usar el programa.
2. Generar el fichero Netlist mediante el programa Eeschema. Con esto obtenemos un archivo con extensión *.net*.
3. Crear la asociación entre los componentes del esquema y los módulos correspondientes del circuito impreso. Para eso utilizamos el programa Cvpcb que emplea la biblioteca *.mod*. Al concluir esta parte generamos dos ficheros con extensiones *.cmp* y *.net* (actualizada).
4. Abrir el programa de Pcbnew y de este modo cargamos y leemos los archivos creados anteriormente. De este modo se cargan todos los módulos y deberemos colocarlos en su posición de acuerdo con las especificaciones de diseño; del mismo modo será necesario crear las pistas correspondientes. Obtendremos un fichero con extensión *.brd*. Aquí crearemos también los ficheros de taladrado y de fototrazado que serán los necesarios para su posterior fabricación.

3.4. Construcción de la placa

La placa se ha fabricado en la propia universidad y fue encargada al departamento de automática. El proceso mediante el cual se fabrica la placa o PCB recibe el nombre de "cnc circuit board routing".

Para su fabricación han sido necesarios los ficheros creados en el apartado anterior, los cuales contienen toda la información acerca del circuito. Incluyen las instrucciones necesarias para taladrar y dar forma a nuestra placa de acuerdo con nuestras especificaciones.

Este proceso de creación de circuitos impresos se realiza en distintas etapas. Partimos de una placa de vidrio a la que se le ponen dos planchas de cobre (una por cada lado del vidrio). Para la creación de la placa se utiliza una máquina de control numérico CNC. Una vez tenemos la superficie básica en la cual se creará nuestra PCB comenzaremos con el proceso de mecanizado de la misma.

- Atacado: en primer lugar retiraremos mediante un taladro todo el cobre de la placa que no hace falta; con lo que iremos retirando todo el material sobrante e iremos dejando el cobre correspondiente a las distintas vías mediante las que irán conectados los distintos componentes.
- Perforado: consiste en realizar los taladrados correspondientes a los agujeros en los cuales irán colocadas las patillas de los distintos componentes electrónicos. Estos taladrados tienen que tener cobre en su diámetro (realizado en la fase anterior) para que los componentes puedan conectarse unos con otros mediante las vías. En el caso de que las capas superior e inferior estén comunicadas, los orificios serán metalizados.
- Máscara antisoldante: es un recubrimiento utilizado para proteger el circuito impreso; de esta manera impedimos que se pueda soldar encima de él y permitir así la soldadura en aquellos lugares en los que está previsto.
- Marcaje de componentes o serigrafía: consiste en imprimir información en la placa. Se utiliza para nombrar los componentes, indicar sus dimensiones, posición, logotipos…; es decir, para referenciar cualquier característica que se considere útil para el circuito. En función del espacio disponible se podrá escribir más o menos información.

A estas fases pertenecientes a la creación del circuito impreso *(Figura 35)*, debemos también mencionar el mecanizado propio que pueda tener nuestra pieza. En este caso nos referimos al fresado de los huecos de las ruedas motorizadas y al perímetro circular de todo el circuito.

Para este primer prototipo del robot, el circuito impreso creado consta de las fases de fresado, atacado y perforado; en este caso no se han realizado las fases de serigrafía y máscara antisoldante.

El no incluir la máscara antisoldante nos creará distintos problemas de los que hablaremos en otro apartado.

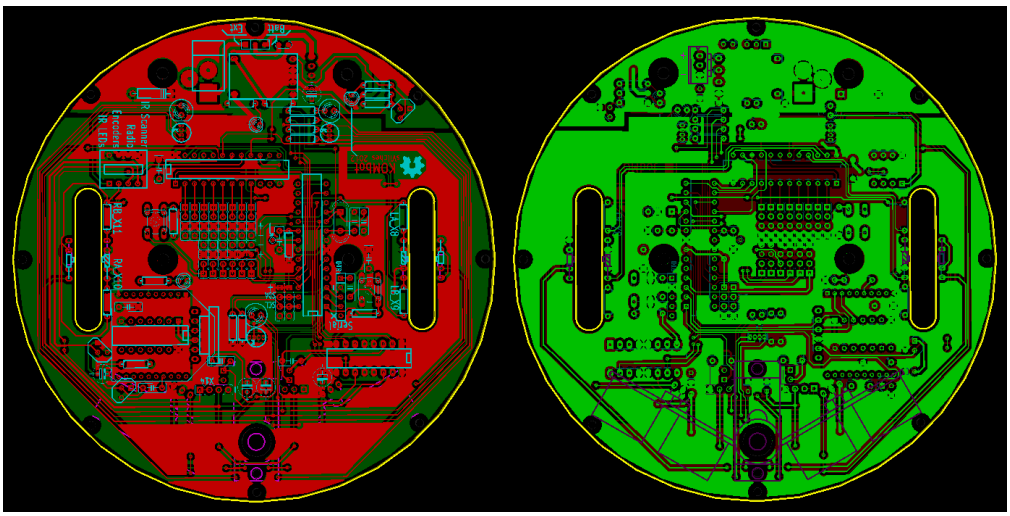


Figura 35: Circuito impreso

4. Diseño mecánico

4.1. Introducción

En esta parte del proyecto vamos a tratar la estructura mecánica del robot y que ha sido construida mediante las impresoras 3D de la universidad.

A la hora de realizar el chasis teníamos que tener en cuenta una serie de especificaciones que se amoldaran a la estructura del robot y permitiese su correcta implementación con el resto de elementos.

Teniendo en cuenta las dimensiones de la placa y la posición de los sensores y demás componentes electrónicos del robot, el chasis tenía de cumplir una serie de condiciones; el soporte imprimible debía contener:

- Motores DC, con holgura mínima, que se puedan colocar y quitar por presión. Una vez dentro no pueden deslizar en ningún sentido.
- Ruedas, que encajen en el eje del motor y sean solidarias, sin cabeceo. El eje podrá salir por el otro lado en el caso de que se necesiten ajustes posteriores. Las ruedas tendrán 16 agujeros con forma de sector circular con las dimensiones del plano "*Ruedas laterales*" del apartado "*Planos referidos al chasis*" del Anexo. Lo óptimo sería que el agujero y de la columna entre uno y otro sean del mismo ancho, para que a velocidad constante la señal que den los encoders sea cuadrada al 50% de ciclo. Por lo tanto, tenemos que evitar que al imprimir las ruedas la primera capa se "achate" y los agujeros salgan más pequeños.
- Soporte para batería, con holgura por si se hincha, pero que no se mueva demasiado de su sitio. Hay que probar si el puente es imprimible. También hay que pensar qué hacer con los cables para que no estorben, y por dónde los conectaríamos a la placa.
- Dos ruedas locas equidistantes
- 4 tuercas empotradas en las posiciones que aparecen en el plano de "*Medidas de la placa*" del apartado "*Planos referidos a la placa base*" del Anexo.

Otras características importantes a tener en cuenta son:

- Tenemos que evitar que el chasis entre dentro de la zona destinada a los sensores.

- Por la parte inferior de la PCB asoman los pines cortados de los componentes. Hay que tener en cuenta ese espacio que hay que dejar libre.
- Comprobar que existen juntas tóricas de ese tamaño (40mm de diámetro exterior).
- Estructura suficientemente rígida, pero con el mínimo plástico necesario.

Todas las dimensiones finales pueden encontrarse de forma más detallada en el apartado correspondiente a los planos. Sin embargo en esta aparte mencionaremos algunas de las medidas más importantes del robot para una mayor compresión de los mismos.

Teniendo claro los objetivos que teníamos que conseguir con el chasis, el siguiente paso sería diseñarlo y crearlo. Para ello usaremos las herramientas que indicaremos a continuación.

Para el diseño y construcción de la estructura hemos usado diversos programas para su realización:

- OOML2
- Qt Creator
- OpenScad
- Replicatorg
- Pronterface & Slic3r

A continuación explicaremos cada uno de estos programas con más detenimiento.

4.2. Programas software

4.2.1. OOML2

La creación de nuestro chasis ha sido posible gracias a la utilización de OOML. El diseño está basado en el uso de esta librería.

Object Oriented Mechanics Library u OOML es precisamente eso, una librería de C++ orientada a objetos mecánicos. Aporta los recursos necesarios para crear objetos, heredando algunas partes de otros y teniendo acceso a cientos de librerías de geometría. Es una facultad C++ que no está sujeta a las limitaciones de OpenScad.

La primera versión de esta librería (OOML) fue desarrollada por Juan González Gómez y Alberto Valero Gómez en la Universidad Carlos III de Madrid. Actualmente existe una segunda versión (OOML2) que es la que se ha usado para este proyecto y que ha sido desarrollado principalmente por Alberto Valero Gómez, Rafa Treviño, Mario Almagro y Nieves Cubo. En la actualidad es Alberto Valero quien se encarga de su mantenimiento.

El diseño de las distintas piezas del chasis está basado OOML. En nuestro caso se ha utilizado como sistema operativo Linux por ofrecer software libre.

Para poder trabajar en el diseño de las piezas, en primer lugar es muy importante tener en nuestro equipo la librería con los ficheros OOML; dichos ficheros se pueden descargar de la página:

http://www.thingiverse.com/thing:17761
http://iearobotics.com/oomlwiki/doku.php?id=start

Una vez en la página nos fijaremos en el índice de la derecha. En la sección de Download & Install encontraremos todo lo necesario para comenzar. Buscamos el enlace, lo descargamos y lo instalamos en nuestro ordenador. El proceso es ligeramente distinto en función del sistema operativo que tengamos. La librería se actualiza continuamente con nuevos contenidos, por lo que el enlace de descarga siempre ofrece la última versión disponible.

OOML nos permite de forma sencilla realizar diseños de distintos elementos en 3D para generar el código OpenScad; tras esto deberemos crear los archivos .stl necesarios para imprimir las piezas con las impresoras 3D.

Mediante figuras geométricas sencillas podremos crear otras más complejas mediante la combinación de éstas. OOML reúne tanto el diseño de piezas a través de código (al igual que OpenScad) con el poder de programación de lenguaje de programación de objetos como es C++.

Debemos aclarar que OOML se encuentra aún en fase beta; sin embargo está dando muy buenos resultados y hasta el momento puede instalarse en sistemas Linux y Mac OS/X.

Es muy importante tener localizada la carpeta con los ficheros de OOML ya que será de vital importancia a la hora de diseñar y programar las distintas piezas de nuestro robot. Deberemos indicar la ruta en la que se encuentra al programa que usaremos para crear el código de los distintos objetos.

OOML tiene una serie de características que ayudan a entender su función:

- OOML es un conjunto de herramientas escritas en C++ que permite a los diseñadores crear distintas piezas mecánicas utilizando el lenguaje C++. Debido a esta característica de utilizar C++, los diseñadores pueden utilizar cualquier otra librería existente de C++ para calcular, manipular y crear sus piezas.
- OOML aplica el modelo de programación orientada a objetos a los diseños mecánicos; esto significa que las piezas pueden ser heredadas unas de otras. Una parte se puede unir otras piezas existentes, y así sucesivamente creando de este modo piezas nuevas a partir de otras previas.
- Hasta este momento OOML permite su uso en cualquier ordenador que tenga un compilador de C++ estándar.
- Como inconveniente tenemos que indicar que hasta la versión actual de OOML genera sólo código .scad. Por lo que para obtener los archivos *.stl* de las distintas piezas creadas es necesario usar OpenScad y desde allí crear los archivos.
- OOML es de código abierto; esto implica poder usarlo, modificarlo y compartirlo. Permite a los diseñadores a crear y compartir sus propias librerías de partes, enriqueciendo así a la comunidad.

En nuestro caso hemos elegido el programa Qt Creator para poder utilizar OOML. Debido a que no podemos ver en 3D las figuras que se van creando, es necesario utilizar OpenScad que sí nos permite mostrarlo; de este modo podemos tener una referencia de lo que vamos creando y así ir puliendo el diseño.

Todas las piezas del chasis han sido creadas gracias a la librería OOML2 que nos ha permitido utilizar un código sencillo para implementar cualquier tipo de pieza.

4.2.2. Qt Creator

Qt Creator es un programa desarrollado por Qt Development Frameworks con dos tipos de licencia: una comercial y otra de tipo open source. Nosotros usaremos la segunda. Permite el desarrollo y la creación de aplicaciones de escritorio y entornos móviles. Se basa en la programación C++ y permite su uso en los sistemas operativos de Windows, Mac y Linux.

Ya dijimos que Qt será el programa que empleemos para poder utilizar OOML y así poder crear nuestro chasis. El aspecto visual *(Figura 36)* del programa es el siguiente:

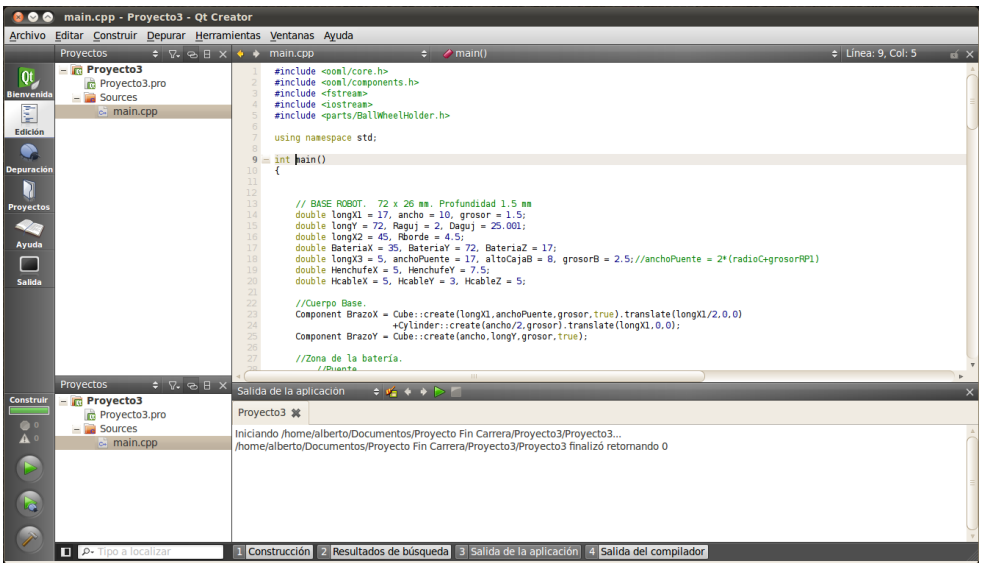


Figura 36: Entorno de Qt Creator

Podemos diferenciar varias partes en el programa.

- En la columna de la izquierda podemos ver todos los proyectos y archivos que se encuentran abiertos en ese momento. Se puede ver a su vez si éstos llevan otros archivos asociados a ellos y donde se encuentran.
- La zona superior central es la parte en la cual trabajamos. Aquí es donde escribimos todo el código de cada uno de los proyectos. Cambiando de archivo en la barra de la izquierda mencionada antes podemos editar cada uno de ellos.
- En la zona inferior es donde se comprueba el código. En el momento de guardar el proyecto y compilarlo (Construir) nos indicará si está bien escrito o si por el contrario tiene errores y donde.

La creación de las distintas partes del chasis se basa en CGS (Constructive Solid Geometry); es decir en operar entre sí distintos solidos simples como esferas, conos, paralelepípedos, etc. Sumando, restando y/o combinando estos elementos podemos crear formas cada vez más complejas.

Finalmente creamos un total de dos archivos distintos: uno para el chasis y otro para las ruedas de los motores. A continuación explicaremos cada uno de ellos.

4.2.2.1. Chasis

En primer lugar iremos explicando el código creado para dar lugar a nuestro chasis. Hemos querido simplificar el diseño todo lo posible y evitar tener que montar diferentes partes. Buscamos crear una pieza sólida y resistente que a la vez fuese lo más elemental para ahorrar costes de material.

A la hora de crear los archivos es importante que incluyamos las librerías *(Figura 37)* pertenecientes a OOML para poder trabajar con ellas en la creación de nuestro código. Para ello debemos indicar en esta parte la ruta en donde se encuentra la librería que hemos descargado con anterioridad y poner cada una de las carpetas de las que consta. Esto es algo común que debemos realizar en todos los archivos que creemos.

```
#include <ooml/core.h>
#include <ooml/components.h>
#include <fstream>
#include <iostream>
#include <parts/BallWheelHolder.h>

using namespace std;
```

Figura 37: Librerías Qt

Para el proceso de construcción del chasis fuimos creando las diferentes piezas poco a poco. En un principio se diseñó un archivo para cada uno de los componentes. Una vez teníamos la forma básica de cada una de las partes del chasis procedimos a unirlas todas en un mismo proyecto final para poder dar la forma definitiva al chasis.

A continuación explicaremos la parte central del código del chasis y veremos sus distintas partes.

4.2.2.1.1. Base y batería

En primer lugar veamos la parte correspondiente a la base y batería del chasis.

```
// BASE ROBOT.  72 x 26 mm. Profundidad 1.5 mm
double longX1 = 17, ancho = 10, grosor = 1.5;
double longY = 72, Raguj = 2, Daguj = 25.001;
double longX2 = 45, Rborde = 4.5;
double BateriaX = 35, BateriaY = 72, BateriaZ = 17;
double longX3 = 5, anchoPuente = 17, altoCajaB = 8, grosorB = 2.5;
double HenchufeX = 5, HenchufeY = 7.5;
double HcableX = 5, HcableY = 3, HcableZ = 5;

//Cuerpo Base.
Component BrazoX =
    Cube::create(longX1,anchoPuente,grosor,true).translate(longX1/2,0,0)
    +Cylinder::create(ancho/2,grosor).translate(longX1,0,0);
Component BrazoY =
    Cube::create(ancho,longY,grosor,true);
//Zona de la batería.
    //Puente
Component Carrilbateria1 =
    Cube::create(longX2,anchoPuente,grosor,true).translate(-(ancho+longX2)/2,0,0)
    +Cube::create(BateriaX+2*grosorB,anchoPuente,BateriaZ,true).translate(-ancho-BateriaX/2,0,(grosor+BateriaZ)/2)
    -Cube::create(BateriaX,anchoPuente+1,BateriaZ,true).translate(-ancho-BateriaX/2,0,BateriaZ/2-1);
    //Paredes laterales
Component Carrilbateria2 =
    Cube::create(longX2,grosorB,altoCajaB,true).translate(-(ancho+longX2)/2,(longY+grosorB)/2,(altoCajaB-grosor)/2)
    +Cube::create(longX2,grosorB,altoCajaB,true).translate(-(ancho+longX2)/2,-(longY+grosorB)/2,(altoCajaB-grosor)/2)
    +Cube::create(grosorB,longY,altoCajaB,true).translate(-longX2-grosorB/2,0,(altoCajaB-grosor)/2);
    //Zona de enganches inferiores
Component Carrilbateria3 =
    Cube::create(longX3,BateriaY,grosor,true).translate(-longX2-longX3/2,0,0)
    +Cylinder::create(Rborde,grosor).translate(-longX2-ancho/2,Daguj,0)
    +Cylinder::create(Rborde,grosor).translate(-longX2-ancho/2,-Daguj,0);
    //Hueco para enchufe 3x6 mm.
Component Henchufe =
    Cube::create(HenchufeX,HenchufeY,2*grosor).translate(-longX2-ancho/2,-Daguj+2*Rborde,0);
    //Hueco para cables de bateria 5x3 mm
Component Hcable =
    Cube::create(HcableX,HcableY,2*grosorB,true).translate(-longX2-grosorB,(longY-grosorB)/2,grosor+HcableZ);
Component Base3 =
    BrazoX + BrazoY + Carrilbateria1 + Carrilbateria2 + Carrilbateria3 - Henchufe - Hcable;
Base3.translate(0,0,grosor/2);
```

Figura 38: Código chasis 1

La creación del chasis comenzó con la creación de dos barras en forma de T; estas se encargarían de dar soporte a los motores y a una de las ruedas locas.

Así mismo se creó la cavidad para albergar la batería; ésta se encuentra unida a la barra en T y su diseño debe permitir la colocación de la otra rueda loca. Hay que tener en cuenta que la batería puede inflarse un poco cuando se conecta, por lo cual en la cavidad hay que considerar una cierta holgura.

El código correspondiente a esta parte se muestra en la imagen anterior *(Figura 38)*.

4.2.2.1.2. Motores

Los motores se sitúan en los extremos del chasis, en la barra en T. Los motores van recogidos en un pequeño cubículo semiabierto de tal modo que se permita su extracción de forma sencilla y que a su vez los mantenga sujetos una vez estén colocados. Debe soportar el movimiento de estos una vez que se pongan en funcionamiento. Fue complicado dar con la forma precisa para ello, pero finalmente tras varios diseños dimos con la forma definitiva cuyo código se puede ver en la siguiente imagen *(Figura 39)*.

```
//SOPORTE MOTORES.
double anchoM1 = 17, largoM1 = 15, altoM1 = 11, radioM1 = 6.5;
double anchoM2 = 17, largoM2 = 1.5, largoM3 = 11, radioM2 = 2, altoM2 = 2.5;
double anchoM3 = 2.5;
double  AltMot = 6.5;

    //Paredes laterales 17 x 15 x 10 mm. Hueco cilindrico Radio 6 mm.
Component MLateral =  Cube::create(anchoM1,largoM1,altoM1,true)
                      -Cylinder::create(radioM1,largoM1+1).rotate(90,0,0);
Component MLateral2 = Cube::create(anchoM3,largoM3,altoM1,true).translate((anchoM1-anchoM3)/2,-(largoM3 +largoM1)/2,0);
Component MLateral3 = MLateral2.mirroredCopy(-1,0,0);//*
//Pared frontal 12 x 1.5 x 10 mm.
Component MFrontal =
    Cube::create(anchoM2,largoM2,altoM1).translate(0,-(largoM3+(largoM1-largoM2)/2),0)
    -Cylinder::create(radioM2,largoM2+1).rotate(90,0,0).translate(0,-(largoM3+(largoM1-largoM2)/2),0)
    -Cube::create(2*radioM2,largoM2+1,(radioM2+altoM1)/2).translate(0,-(largoM3+(largoM1-largoM2)/2),-altoM1/2+radioM2);
    //Tope trasero 17 x 1.5 x 2.5 mm.
Component MTrasero = Cube::create(anchoM1,largoM2,altoM2,true).translate(0,(largoM2+largoM1)/2,(altoM1-altoM2)/2);
    //Unión. Altura de colocación del Motor.
Component AlturaMotor = Cube::create(anchoM1,largoM1+largoM2+largoM3,AltMot).translate(0,largoM2/2-largoM3/2,(altoM1+AltMot)/2);
    //Eliminación zona lateral
Component Elimin =
    Cube::create(anchoM1/2,largoM1+largoM2+largoM3,radioM1+altoM1/2).translate(anchoM1/3,largoM2/2-largoM3/2,(radioM2-altoM1)/2-1.5);
Component MotorIzda = MLateral + MFrontal + MTrasero + AlturaMotor + MLateral2 + MLateral3 - Elimin;
MotorIzda.rotate(180,0,0).translate(0,(longY-largoM1)/2+largoM2-largoM3,AltMot+altoM1/2);

Component MotorDcha = MotorIzda.mirroredCopy(0,-1,0);
Component Motores = MotorIzda + MotorDcha;
```

Figura 39: Código chasis 2

4.2.2.1.3. Ruedas locas

Esta parte se refiere a las ruedas locas del robot. No son ruedas propiamente dichas; en realidad son dos canicas colocadas en los extremos de la base que dan estabilidad a toda la base manteniéndola nivelada. Para poder sujetar estas canicas se han diseñado unos soportes con forma cilíndrica que permiten a su vez libertad a la hora de girar en

cualquier dirección. Son un total de dos distintas, cada una de una altura distinta debido a la posición que ocupan en el chasis. Esto lo podemos ver a continuación *(Figura 40).*

```
//RUEDA LOCA CON CANICA 12.7 MM (ZONA LIBRE).
double radioC = 7, grosorRP1 = 1.5, altoRP1 = 28;
double largoRPY1 = 6.5, altoRPZ1 = 3, altoRPZ2 = 3;
    //Cilindro central.
Component BaseRP1 =
    Cylinder::create(radioC+grosorRP1,radioC+grosorRP1,altoRP1,100,false).translate(0,0,-2*grosorRP1);
Component BaseRP2 =
    Cylinder::create(radioC-grosorRP1,radioC-grosorRP1,altoRP1-grosorRP1,100,false).translate(0,0,-2*grosorRP1);
Component BaseRP3 = Cylinder::create(radioC+grosorRP1,radioC+grosorRP1,grosor,100,false).translate(0,0,radioC);
    //Dimensión de la canica.
Component Canica = Sphere::create(radioC);
    //Huecos de holgura para entrada de canica.
Component HuecoRP1 = Cube::create(2*(radioC+grosorRP1)+1,largoRPY1,altoRPZ1+1,true).translate(0,0,-(altoRPZ1+1)/2);
Component HuecoRP2 = Cylinder::create(largoRPY1/2,0,altoRPZ2).translate(radioC+grosorRP1/2,0,altoRPZ2/2);
Component HuecoRP3 = HuecoRP2.mirroredCopy(-1,0,0);
Component HuecoRP4 =
    Cube::create(2*(radioC+grosorRP1),largoRPY1/2,5*radioC/4,false).translate(-(radioC+grosorRP1)-1/2,-largoRPY1/4,-2*grosorRP1);
Component RuedaP1 = BaseRP1 - BaseRP2 + BaseRP3 - Canica - HuecoRP1 - HuecoRP2 - HuecoRP3 - HuecoRP4;
RuedaP1.rotate(180,0,90).translate(longX1,0,altoRP1-2*grosorRP1);

// RUEDA LOCA CON CANICA 12.7 MM (ZONA BATERIA).
double altoRP2 = 11;
Component BaseRP1b = Cylinder::create(radioC+grosorRP1,radioC+grosorRP1,altoRP2,100,false).translate(0,0,-2*grosorRP1);
Component RuedaP2 =BaseRP1b - BaseRP2 - Canica - HuecoRP1 - HuecoRP2 - HuecoRP3 - HuecoRP4;
RuedaP2.rotate(-180,0,90).translate(-ancho-BateriaX/2,0,BateriaZ+altoRP2-grosorRP1-grosor);
```

Figura 40: Código chasis 3

4.2.2.1.4. Agujeros y suma de elementos

La última parte del chasis corresponde con la creación de los agujeros de soporte. Existen un total de cuatro agujeros (dos en la parte de los motores y dos en la zona de la batería). Estos orificios son muy importantes ya que son los que permiten al chasis unirse a la placa mediante tornillos.

Finalmente y con todos los elementos creados, es necesario sumarlos todos para conformar la forma final del chasis *(Figura 41).*

```
//AGUJEROS DE ENGANCHE A LA PLACA.
Component Agujero1 = Cylinder::create(Raguj,10*grosor).translate(0,Daguj,0);
Component Agujero2 = Agujero1.mirroredCopy(0,-1,0);
Component Agujero3 = Cylinder::create(Raguj,4*grosor).translate(-longX2-ancho/2,Daguj,0);
Component Agujero4 = Agujero3.mirroredCopy(0,-1,0);

//AGUJEROS PARA TUERCAS EN ENGANCHE MOTORES
double radioT = 3.5, alturaT = 2;
Component Tuerca1 = Cylinder::create(radioT,AltMot).translate(0,Daguj,alturaT+AltMot/2);
Component Tuerca2 = Tuerca1.mirroredCopy(0,-1,0);
Component Tuerca3 = Cylinder::create(radioT,AltMot).translate(-longX2-ancho/2,Daguj,3*alturaT/4+AltMot/2);
Component Tuerca4 = Tuerca3.mirroredCopy(0,-1,0);

//SUMA DE ELEMENTOS
Component Chasis1 = Base3 + Motores + RuedaP1 + RuedaP2;
Component Chasis = Chasis1 - Agujero1 - Agujero2 - Agujero3 - Agujero4 - Tuerca1 - Tuerca2 - Tuerca3 - Tuerca4;
```

Figura 41: Código chasis 4

A la hora de guardar los proyectos, debemos crear además un archivo (o script) que indique al programa de OpenScad las características de nuestra pieza y cuya extensión

debe ser ".scad". De esta forma podremos contemplar una visión en 3D de las piezas. En la siguiente imagen *(Figura 42)* podemos verlo.

```
IndentWriter writer;
writer << Chasis;

//Guardar código en archivo OpenScad;
ofstream file("Proyecto3.scad");
file << writer;
file.close();
```

Figura 42: Código de creación .scad

Esto es sumamente útil; ya que nos permite visionar nuestras piezas para ir depurando el código y corregir los posibles errores de dimensionado, forma…

4.2.2.2. Rueda

Las ruedas debido a que se colocan directamente sobre el eje de los motores, no tenemos más remedio que imprimirlos a parte.

```
int AgujerosEncoder = 8;

Component CuerpoRueda = Cylinder::create(20, 4, 100, true);
Component AgujeroLlanta = Toroid(20, 1, 100);

Component AgujeroEje = Cylinder::create(9, 4, 100, true);
AgujeroEje = AgujeroEje.translate(0, 0, 1);

Component TornilloGr = Cylinder::create(2, 6);
TornilloGr = TornilloGr.translate(0, 0, -1.5);

Component TornilloPeq = Cylinder::create(1.5, 6);

Component AgujerosTorn = TornilloGr;
TornilloPeq = TornilloPeq.translate(6.4, 0, 0);
for (int i = 0; i < 4; i++ )
    AgujerosTorn = AgujerosTorn + TornilloPeq.rotatedCopy(0, 0, 90*i);

Component CirculoExt = Cylinder::create(16.5, 4, 100, true);
Component CirculoMed = Cylinder::create(13.5, 4, 100, true);
Component CirculoInt = Cylinder::create(10.5, 4, 100, true);

Component AnilloExt = CirculoExt - CirculoMed;
Component AnilloInt = CirculoMed - CirculoInt;
Component AgujeroAnillo = CirculoExt - CirculoInt;

Component CuboAux = Cube::create(50, 50, 10, true);
CuboAux = CuboAux.translate(0, -25, 0);
Component CuboAux2 = CuboAux.scaledCopy(1);
CuboAux2 = CuboAux2.rotateAround(0,0,180-(180/AgujerosEncoder),0,0,0);
Component PiezaAux = CuboAux + CuboAux2;

Component VentanaExtAg = AnilloExt - PiezaAux;
Component VentanaIntAg = AnilloInt - PiezaAux;
VentanaIntAg = VentanaIntAg.rotate(0,0, (-90/AgujerosEncoder));
Component VentanaAguj = VentanaExtAg + VentanaIntAg;
Component VentanasAnillo = VentanaAguj;
for (int i = 0; i < AgujerosEncoder; i++)
    VentanasAnillo = VentanaAnillos + VentanaAguj.rotatedCopy(0, 0, 360*i/AgujerosEncoder);

Component Rueda = CuerpoRueda - AgujeroLlanta - AgujeroEje - AgujeroAnillo - AgujerosTorn;
Rueda = Rueda + VentanasAnillo;
```

Figura 43: Código ruedas

Es importante tener en cuenta en su diseño que deben tener una cavidad a lo largo de la superficie de apoyo con el suelo para poder colocarles una goma y que de este modo a la hora de girar no resbale por las superficies.

También debe tener en las bases unas perforaciones concéntricas a modo que atraviesen la pieza; de esta forma podemos utilizar los encoders para que regulen la velocidad del robot en función de lo rápido que detecten estas "ventanas".

Igualmente su diseño debe contener los agujeros necesarios para poder unirlas a los motores.

En la imagen anterior se puede apreciar su código *(Figura 43).*

4.2.3. OpenScad

OpenScad es un programa enfocado a la creación y representación 3D de los elementos creados mediante el lenguaje C++.

Este programa es de software libre y ha sido desarrollado y permite su uso en todos los sistemas operativos.

Como hemos indicado antes OpenScad permite la recreación de figuras creadas a partir de un script. Es una herramienta sumamente útil ya que nos permite ver con todo detalle las piezas. Podemos verlas desde todos los ángulos, así como sus superficies y ejes.

El entorno de este programa es el siguiente *(Figura 44)*.

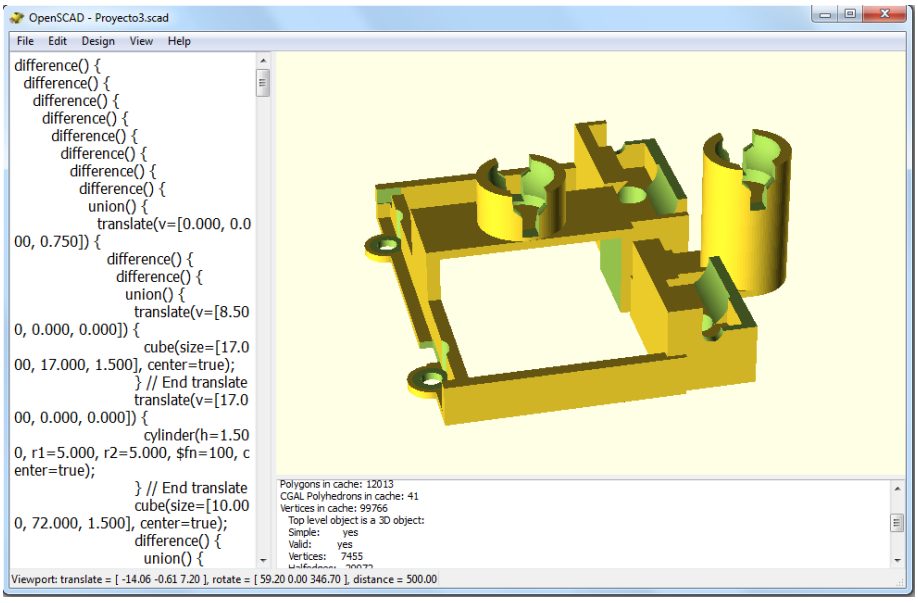


Figura 44: Entorno OpenScad

Podemos ver tres ventanas distintas:

- En la columna de la izquierda tenemos el editor. Aquí es donde podemos crear las figuras directamente mediante módulos. Pondríamos las instrucciones de forma similar a como hemos hecho en Qt Creator y lo iríamos componiendo.
- En la columna superior derecha nos encontramos con la ventana de la vista preliminar; es decir es la parte del programa en el que podemos ver en 3D cómo se va componiendo nuestra figura.
- La columna inferior derecha es la que nos indica que operaciones realiza el programa y los posibles errores si los hubiera.

Como ya tenemos realizados los archivos, lo único que haremos será cargarlo.

Para que nuestro archivo sea imprimible primero deberemos compilarlo y renderizarlo, y a continuación, se exporta el archivo a STL para que pueda ser reconocida por la impresora. En nuestro caso crearemos dos archivos distintos: uno referente al chasis y otro a la rueda. Cada uno de estos archivos deberemos generarlos en STL para poder imprimir nuestras piezas.

4.2.4. Replicatorg

Replicatorg es otro programa es de software libre. Será el último que usaremos para conformar el chasis del robot y para su uso es necesario tener una impresoras 3D como las que se encuentran en el laboratorio de la universidad.

El programa de Replicatorg tiene una interfaz *(Figura 45)* dividida en tres zonas distintas:

- En primer lugar arriba del todo hay una fila con una serie de botones. Dichos botones son los que nos permiten controlar la impresora para que comience a imprimir.
- La zona central es donde aparece todo lo relativo a nuestro archivo. Una vez se cargue, aquí será donde podamos editar sus propiedades para adecuarlas a nuestras necesidades.
- Finalmente en la zona inferior se encuentra la parte relativa a la información. Todos los movimientos que vayamos haciendo mientras que el programa se encuentre operativo se verán reflejados aquí.

También es importante mencionar la barra de File, Edit, G-Code y Machine, puesto que mediante estas opciones podremos cargar o abrir archivos, construir el código para que pueda imprimirse o bien abrir el panel de control de la máquina para poder controlar los distintos elementos de la impresora (extrusor, base…) entre otros.

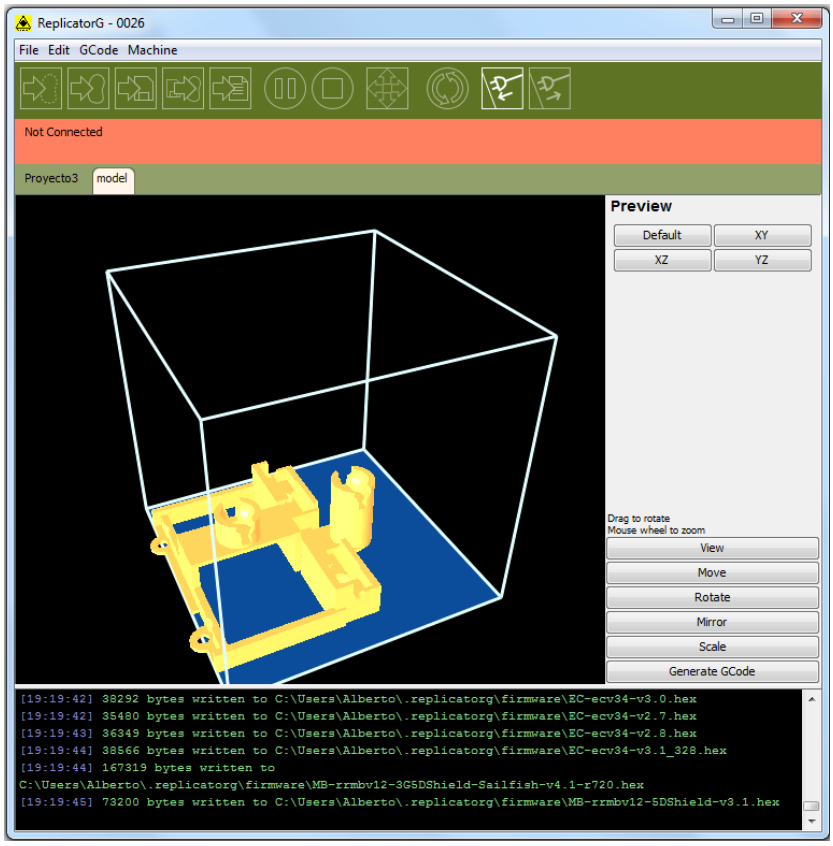


Figura 45: Entorno Replicatorg

Para comenzar a trabajar con el programa debemos cargar nuestro archivo de extensión *.stl* que hemos creado anteriormente en el programa OpenScad.

La ventana central está dividida a su vez en dos partes:

- En la parte izquierda podemos ver una vista previa de nuestra pieza. Si nos fijamos la pieza se encuentra posicionada dentro de un cubo; este cubo representa los márgenes de impresión de nuestra impresora. Si alguna parte de la figura no se encuentra dentro de estos márgenes no se imprimirá; por lo tanto deberemos ajustar la posición de la pieza para que quepa dentro. Gracias a esta vista previa podemos hacernos una idea de su posición y su tamaño.
- En la parte derecha nos encontramos con una serie de botones para modificar la pieza. Estos ajustes se verán reflejados en la ventana izquierda permitiendo así que la pieza se encuentre en la posición deseada para imprimir.

Tras haber cargado el archivo deberemos posicionar correctamente la pieza para que pueda ser impresa por la impresora. Usamos para ello los botones mencionados anteriormente.

Es importante conectar la impresora y habilitar esta conexión con el programa para que puedan comunicarse.

Posteriormente mediante la opción de "Generar GCode" se creará el código que interpreta la impresora. De este modo nos saldrá otra ventana en la que se podrán configurar las distintas opciones de impresión (habilitado por el botón Print-O-Matic).

- Settings: aquí nos encontramos con la posibilidad de modificar la forma en la que se crea nuestra pieza (% de relleno, nº de pasadas o velocidad del extrusor).
- Plastic: esta pestaña hace referencia a las propiedades del filamento de plástico que se fundirá para dar lugar a las distintas piezas
- Extruder: se refiere al extrusor de la máquina por donde sale el plástico fundido y nos permite indicar las propiedades del mismo.

Nota: Para poder continuar y que podamos generar el GCode, será necesario tener instalado Python, que no es más que un lenguaje de programación multiplataforma que soporta orientación a objetos, programación imperativa y en menor medida programación funcional. Para instalar la versión que mejor se adapte a nuestro sistema operativo seguiremos este enlace:

http://www.python.org/download/

Normalmente las opciones que se suelen modificar son las pertenecientes a la pestaña "Settings"; ya que las otras dos son específicas del tipo de cordón de plástico y extrusor de nuestra impresora.

Una vez seleccionadas las opciones que mejor nos convenga crearemos el archivo *.gcode* con el mismo nombre que el archivo cargado y que contiene todos los datos hasta ahora, así como las nuevas modificaciones que hayamos podido hacerle. Este nuevo archivo lo crea en la misma carpeta en la que tenemos el archivo *.stl*.

Tras un tiempo que variará en función de la complejidad de la pieza, el programa crea un archivo con la trayectoria que debe seguir la impresora para poder imprimir la pieza seleccionada.

Lo único que nos queda ahora es pulsar el botón de "Build to file" perteneciente a la fila de botones que se encuentra en la zona superior del programa. De este modo la impresora comenzará a funcionar.

La impresora comienza a calentar el plástico para fundirlo; del mismo modo también comienza a calentar la superficie en la cual se apoya la pieza. Se puede ir comprobando desde el propio programa como la temperatura de ambas partes comienza a aumentar. Cuando alcance la temperatura óptima la impresora se pondrá en marcha y comenzará a crear la pieza seleccionada por capas.

Una vez terminada la pieza solo nos queda retirarla de la plataforma de la impresora.

4.2.5. Pronterface & Slic3r

La última impresora en añadirse al laboratorio fue una del tipo Prusa Mendel. Para imprimir las piezas con este modelo se ha usado un conjunto de dos programas: Pronterface y Slic3r.

Esta impresora es de una mayor dimensión, lo que nos permite crear piezas más grandes. Además gracias a estos programas podemos imprimir varias piezas a la vez si fuese necesario, con lo que nos ahorramos tiempo.

Del mismo modo que Replicatorg, Pronterface utiliza el lenguaje de programación de Python; por lo que será necesario igualmente tenerlo instalado en el ordenador.

Para poder utilizar Pronterface deberemos descargar y descomprimir el archivo en el que se encuentra el programa. Podemos acceder a él mediante el siguiente enlace:

https://github.com/kliment/Printrun/archive/master.zip

Para instalar Slic3r, iremos a su página oficial.

Veamos cómo debemos usar estos programas para poder imprimir con esta impresora. Podemos comprobar que nos encontramos ante un conjunto de dos programas separados.

- El programa principal es Pronterface; es el que nos permite ver y controlar el estado de la impresora en cada momento.
- Slic3r por el contrario nos permite crear el archivo *.gcode* que usará Pronterface para imprimir las piezas.

Abrimos Pronterface y se nos abre una ventana como la siguiente *(Figura 46)*.

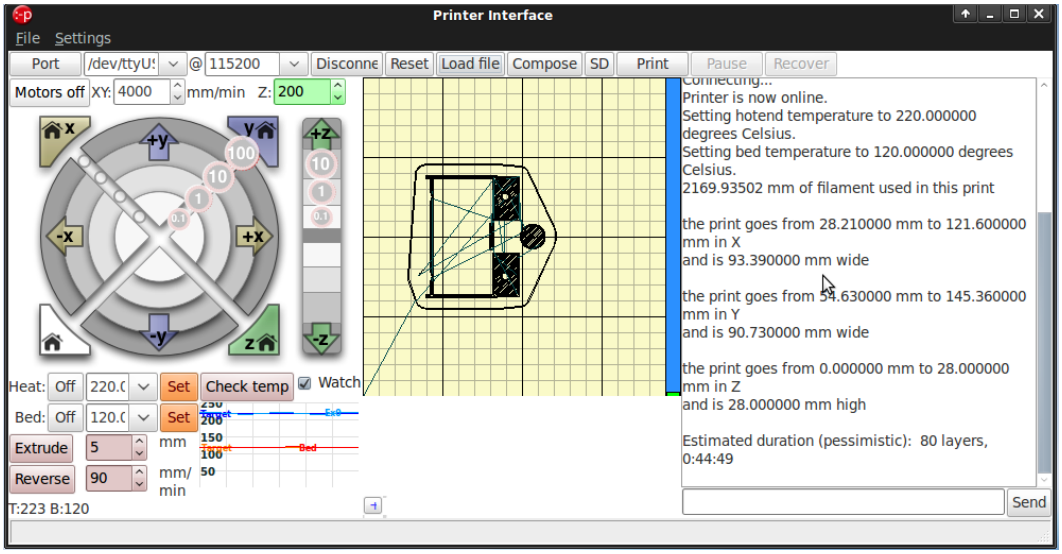


Figura 46: Entorno Pronterface

El interfaz del programa se divide en varias partes:

- Panel de control: situado en la parte superior izquierda. Nos permite mover libremente la impresora a las distintas posiciones para un mejor manejo (retirar piezas, manipulación, etc.)
- Temperaturas y extrusor: está debajo del panel de control. Aquí podemos ver la temperatura a la que se encuentra el extrusor y la cama en la que se apoyan las piezas, así como establecer su valor.
- Gráfica: se encuentra en medio de la interfaz. Es una cuadrícula que muestra de forma visual en cada momento el avance de la impresora. Pudiendo saber en qué capa de impresión vamos.
- Información: está en la parte de la derecha. Se encarga de reflejar en todo momento la información de cada una de las acciones que estamos realizando.

Tras conectar la impresora y realizar la comunicación con el programa es necesario calentar el extrusor y la cama para así comenzar a imprimir. Una vez alcancen la temperatura óptima será el momento de cargar el GCode; esto lo crearemos con el programa de Slic3r *(Figura 47)*.

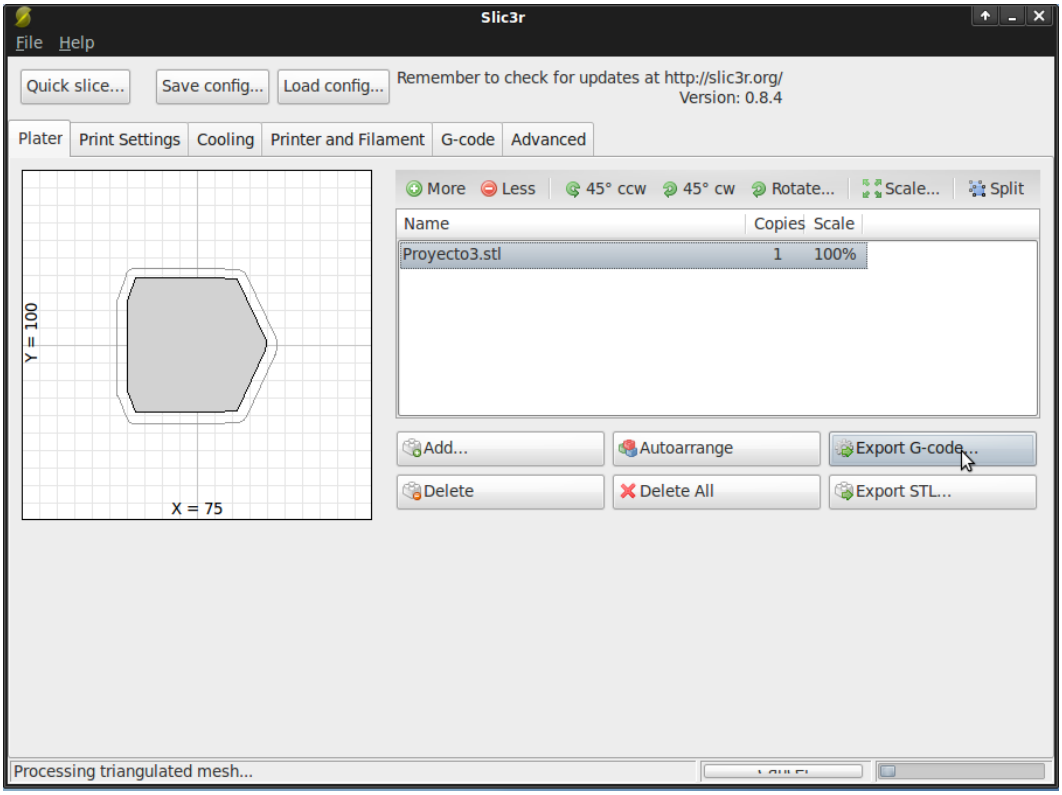


Figura 47: Entorno Slic3r

Lo primero que deberemos hacer será cargar una configuración de impresión (tales como grosor del hilo, capas interiores, altura entre capas…). Si no se tiene ninguna previa podemos crearla y guardarla para posteriores usos.

Una vez tengamos la configuración cargada deberemos añadir el archivo .stl que queramos imprimir. En la zona izquierda del programa se puede ver el tamaño que ocuparía la pieza en la impresora. Además podemos configurar nuestra impresión de tal forma que podamos incluir más piezas si aún tenemos espacio en la impresora.

Cuando estemos satisfechos con la configuración a imprimir crearemos el archivo de extensión GCode para así poder usarlo en Pronterface.

Ahora solo nos queda cargar el archivo en Pronterface y realizar la impresión de la/s pieza/s.

Una vez la pieza esté terminada, deberemos desconectar el extrusor y la base para que se enfríen y así podamos retirar la pieza. En el caso de querer imprimir otra volveríamos a realizar el mismo proceso; si no, procederemos a desconectarlo.

4.3. Impresoras 3D

A día de hoy existen multitud de impresoras 3D de distintos tipos. Su uso en la actualidad es variado: desde prototipos hasta piezas finales. Permiten crear objetos a partir de diseños tridimensionales generados por ordenador (archivos CAD).

Esta clase de impresoras son prácticamente de uso industrial; sin embargo existe otro tipo de impresoras de bajo coste y que permite su uso de forma privada. Debemos destacar además que este grupo de impresoras se caracterizan por seguir una filosofía Open Source mediante la cual cualquiera puede usar y modificar diseños y programas que los diferentes usuarios dejan para compartirlos con la comunidad.

Todos estos modelos de impresoras 3D basan su funcionamiento de impresión en un extrusor de termoplásticos mediante el cual un hilo de plástico es calentado hasta un punto cercano al de fundición y posteriormente es extruido de forma continua para poder dar lugar a las diferentes piezas añadiendo capas de este material hasta conformar la pieza final.

En el laboratorio de la UC3M contamos con dos impresoras de Makerbot Industries y son las que nos han permitido la construcción de las piezas del chasis de nuestro robot. Dichas impresoras son:

- MADRE
- PADRE (a.k.a UC3-PO)

MADRE *(Figura 48)* fue la primera impresora que se construyó en la universidad. Pertenece a la Asociación de Robótica de la UC3M (ASROB). El proyecto surgió a raíz de la propuesta del profesor Juan González Gómez quién colaboró estrechamente con los alumnos que participaron en su construcción. El precio de la impresora fue de 920 € y contó con un presupuesto de 1200 €

Esta impresora es representativa, puesto que fue la primera en construirse y formó parte de un proyecto llamado Clone Wars. Dicho proyecto consistía en conseguir una impresora 3D con el objetivo de replicar otras impresoras del modelo Prusa Mendel. MADRE fue la encargada de imprimir las piezas necesarias de las que surgirían posteriormente diversos “hijos”.

PADRE *(Figura 49)* por el contrario fue la primera impresora comprada por el Departamento de Ingeniería de Sistemas y Automática. En este caso el impulsor fue el profesor Alberto Valero Gómez. Esta impresora tiene un extrusor más fino que la anterior; por lo que nos permite imprimir piezas más precisas y con mayor calidad.

Figura 48: Impresora Madre

Figura 49: Impresora Padre

Gracias al funcionamiento de ambas impresoras hemos podido imprimir varias piezas simultáneamente. Aunque también debemos destacar que las impresoras no han estado siempre operativas; debido a que tienen un gran uso por parte de los alumnos y demás personas con acceso a ellas. En ocasiones nos hemos encontrado con que alguna de las impresoras no funcionaba correctamente, por lo que solo podíamos imprimir piezas con una de ellas.

En 2013 una nueva impresora formó parte del laboratorio. Se trataba de un modelo Prusa de Mendel *(Figura 50)* que proporcionó el alumno y encargado de las impresoras Marco Esteban Illescas y que ha sustituido a la impresora MADRE. Esta impresora ha sido construida por el propio Marco y puso sus servicios a aquellos alumnos que lo requiriesen.

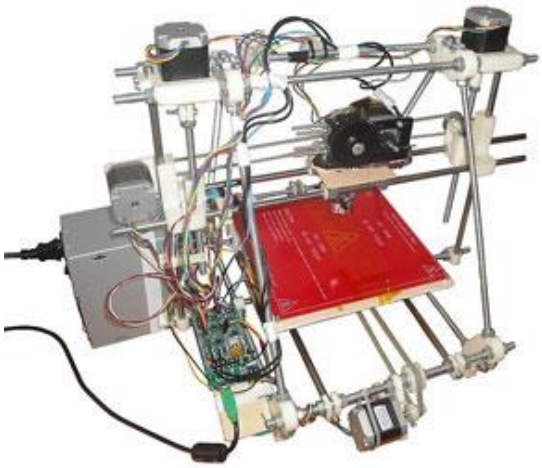


Figura 50: Impresora Prusa Mendel

La aparición de este modelo en el laboratorio ha supuesto muchas ventajas, ya que aparte de ser más rápida que los otros modelos también permite la creación de varias piezas distintas a la vez y con una calidad mejor. También debemos destacar que esta

impresora tiene una base y unas dimensiones mayores, lo que supone poder imprimir unas piezas más grandes (aunque en este caso no fuese necesario).

- ***Bobina de plástico ABS negro***

Todas las impresoras 3D necesitan un hilo de plástico como materia prima para las construcciones que se realicen con ellas. Éstas se venden en bobinas y existe una gran variedad de ellas, tanto en colores como en dimensiones.

• Diámetro del hilo	3 mm
• Material	Plástico ABS
• Color	Negro
• Tamaño	2.3 Kg
• Temperatura de fusión	200 ° C
• Temperatura del extrusor	220° C / 260 ° C
• Temperatura de la cama	60 ° C

En el laboratorio cada impresora tenía una bobina distinta de un color diferente. En función de la impresora usada, las piezas tendrían un color u otro.

La duración del hilo de cada bobina suele ser elevada, ya que se venden por kilos. Nuestro chasis es una pieza pequeña y que tarda relativamente poco en construirse, por lo que gasta muy poco material y permite realizar una gran cantidad de ellas con la misma bobina.

El peso aproximado de todas las piezas impresas es de unos 25 g aproximadamente. Esta es más o menos la cantidad de material que hemos usado y con este dato podemos saber el gasto económico que supone cada pieza.

4.4. Especificaciones de las piezas

En esta parte del proyecto vamos a dar una serie de especificaciones técnicas sobre cada una de las piezas impresas de las que consta el chasis para facilitar su comprensión. Indicaremos el porqué de su forma y como se colocan cada uno de los elementos para su montaje final.

Para un mayor detalle es recomendable fijarse en los planos de las piezas en la sección del anexo.

Las dimensiones de todas las piezas corresponden con la versión imprimida considerando una cierta tolerancia debido al cordón de plástico de la impresora.

4.4.1. Chasis

Como hemos comentado en otras ocasiones, el chasis se ha creado de la combinación de cuatro partes distintas. La unión de todos ellos da lugar a la pieza única del chasis *(Figura 51)*.

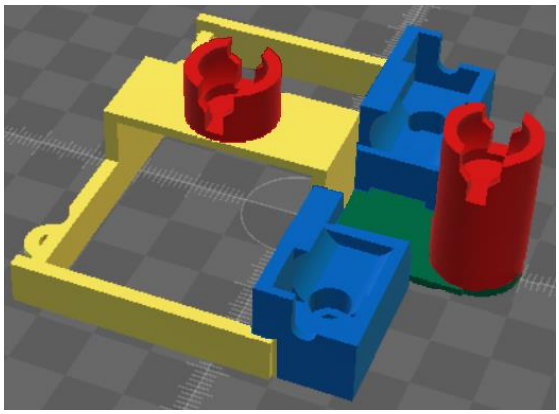


Figura 51: Vista chasis

En la imagen anterior podemos distinguir por colores cada una de estas partes: base (en verde), batería (en amarillo), cavidad de motores (en azul) y ruedas locas (en rojo). A continuación iremos explicando mejor las distintas partes de las que consta.

4.4.1.1. Base

La base del chasis consta de dos perfiles rectangulares unidos formando una T. estos perfiles tienen un grosor de 2 mm y una superficie de 11x75 mm el perfil más largo y 73x18 mm el perfil más corto.

En el perfil largo existen dos agujeros en la zona de los extremos que corresponden con los tornillos delanteros a los que se sujetará la placa al chasis. Encima de ellos se

sitúan las cavidades para los motores. Así mismo el perfil largo se une a la parte de la batería.

El perfil más corto incluye en su extremo de una semicircunferencia de 9 mm de radio y de igual grosor que el resto del perfil y que pertenece a la posición de la rueda loca delantera.

4.4.1.2. Batería

La parte de la batería consta de una caja hueca de dimensiones de 74x36x17 mm. Tenemos que tener en cuenta que aloja una batería de 70x35x15 mm; dicha batería en su funcionamiento puede hincharse unos mm por lo que el hueco tiene en cuenta esta dilatación.

Para utilizar la menor cantidad de material posible pero que a la vez fuese sólido, esta "caja" consta de un perímetro exterior de 3 mm de grosor y una altura de 8 mm. En la parte central tenemos un puente de igual grosor y de ancho 18 mm; este puente nos permite colocarle encima la otra rueda loca al mismo tiempo que delimita la altura de la caja en la cual se sitúa la batería.

Otros detalles de esta parte son los agujeros para los tornillos traseros que se encuentran en la zona final de la batería y que se unen a esta mediante un perfil rectangular. Para una correcta colocación deberemos introducir la batería antes de atornillar el chasis a la base.

También existen dos cavidades rectangulares que permiten el paso de otros elementos del robot y que son necesarios para evitar superposición de los mismos con el chasis. Una correspondiente a los cables de la batería y otro al enchufe de ésta con la placa.

4.4.1.3. Cavidad de motores

Los motores se colocan en los extremos del perfil más largo de la base. Las dimensiones de su "caja" son 17x28x15.5 mm. La zona interior está divida en dos partes diferentes debido al diseño de los motores a los que contienen: una zona cilíndrica y otra cúbica. El grosor de las paredes es variable; este relleno corresponde a la diferencia de la dimensión total y la dimensión del motor.

La "caja" se encuentra abierta parcialmente y los motores se encajan por esta zona gracias a que se pueden introducir ejerciendo una pequeña presión en la zona cilíndrica y que permite su sujeción mediante un enganche que tiene en la parte superior.

Al estar situados en los extremos de la base, deberán tener también los agujeros para los tornillos. Cabe decir que antes de colocar los motores en el chasis, deberemos atornillar el chasis a la placa.

4.4.1.4. Ruedas locas

La cavidad de las canicas que actúan a modo de ruedas pequeñas y que dan estabilidad al chasis una vez están todas colocadas es una cavidad con forma cilíndrica. El extremo en el que se coloca la canica las paredes se adaptan a su forma esférica.

Dicho cilindro está hueco en su parte interna y sus paredes tienen un grosor de 2 mm. El radio de la base es de 9 mm. La altura es distinta para cada rueda: la rueda delantera se encuentra en el extremo del perfil más corto de la base y tiene una altura total de 28 mm incluyendo la base en la que se apoya; mientras que la trasera que se localiza encima del puente de la batería tiene una altura de 12 mm contando con el grosor del puente.

Las canicas se introducen encajándolas haciendo presión en la parte de la cavidad en la cual van colocadas. Aunque el cilindro sea hueco, tiene una base en la zona en la que va la canica y que sirve como apoyo, impidiendo que se introduzca más de lo necesario. En la rueda de la parte de la batería esto no es necesario, puesto que el puente le sirve de apoyo.

- ***Canicas***

Las canicas metálicas son las que permitirán al robot estabilizarse junto con las ruedas de los motores.

• Diámetro	13 mm
• Material	Metal

4.4.2. Ruedas

Son las que se colocan en los motores. Tienen un grosor de 4 mm y un diámetro de 40 mm. A lo largo de su perímetro tenemos un surco central de 1 mm de diámetro que permite la colocación de una goma para que a la hora de mover el robot, éste no resbale en su recorrido.

A modo de radios, las ruedas *(Figura 52)* presentan doce orificios segmentados colocados equidistantes del centro y separados el mismo ángulo entre ellos. Estos orificios constan de dos secciones (una superior y otra inferior) que se encuentran desfasadas una distancia igual a la mitad de su ancho una de otra. Esto nos permite que el robot detecte estos agujeros mediante los encoders, lo que nos permitirá regular el movimiento de los motores y de este modo, controlar la dirección del robot.

En el centro de la rueda tenemos cinco orificios: uno central por el cual se introduce el eje del motor y otros cuatro colocados en forma de cruz que nos permitirán fijar la

rueda al motor de forma más eficiente gracias a un adaptador que incluye el motor. De esta forma la rueda estará mejor sujeta y tendrá una mayor estabilidad cuando el motor esté funcionando.

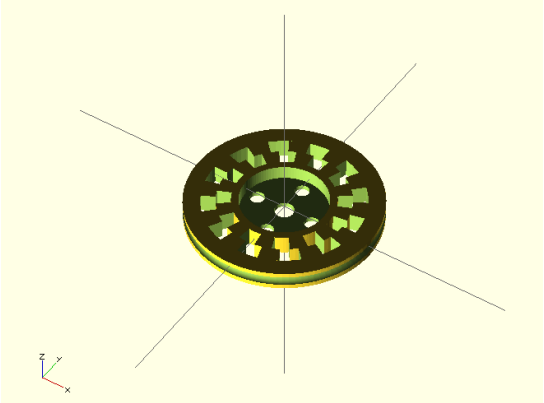


Figura 52: Vista Ruedas

- ***Par de ejes de montaje universales de aluminio Pololu***

Estos son los ejes que irán atornillados a las ruedas *(Figura 53)* y a su vez se unen a los motores. Sus características son:

• Agujero de roscado	4 x 40
• Diámetro eje motor	3 mm

Figura 53: Ejes de montaje de ruedas

- ***Gomas ruedas***

Estas gomas se colocan alrededor de las ruedas permitiendo que el robot tenga un mayor agarre y evitar así deslizamientos a lo largo del recorrido. Características:

• Sección	2 mm
• Diámetro eje motor	40 mm

5. Diseño software

5.1. Introducción

Aquí nos encontramos con el tercer gran bloque del que consta el proyecto. En esta parte nos encargaremos de crear y configurar la parte software del robot de tal manera que éste sea plenamente funcional y se mueva por sí mismo.

Para este apartado del proyecto se ha utilizado como herramienta fundamental el programa "Arduino". Como hemos indicado en otras partes del proyecto Arduino es la plataforma en la cual se basa gran parte del robot. Incluyendo el microprocesador y el entorno de desarrollo.

Arduino es de tipo open-hardware; una gran ventaja en nuestro caso, ya que tanto su diseño como su distribución es libre. No necesitamos de una licencia para poder utilizarlo y existe una gran comunidad con infinidad de proyectos disponibles para todo aquel que esté interesado.

Nos centraremos en el entorno de desarrollo. Veremos cómo funciona el programa de programación de Arduino; así como las distintas líneas de código que se han incluido en el robot para su funcionamiento.

Para que el robot sea considerado plenamente funcional, deberemos cumplir una serie de objetivos que desarrollaremos en mayor profundidad a lo largo de toda esta sección:

- Lectura de distancias de los sensores US.
- Lectura del sensor IR.
- Control del ángulo girado por el IR a través de un miniservo.
- Control en velocidad de los motores de continua.
- Odometría (detección del ángulo girado por cada rueda).
- Mapeado.

5.2. Arduino

En este apartado veremos un poco el programa de Arduino que será el utilizado para programar todo el robot.

Ya hemos mencionado que Arduino es una plataforma abierta que nos permite crear prototipos mediante un hardware y software flexible y sencillo de usar. Fue creado con la intención de que todo aquel que esté interesado en crear objetos y entornos interactivos pudiera hacerlo de la manera más fácil posible.

Arduino toma la información del entorno que le rodea mediante los pines de entrada a los que pueden ir conectados multitud de sensores y actuadores diferentes.

Las placas pueden comprarse ya montadas de fábrica o crearlas a mano. Todo el software es libre y se pueden descargar los ficheros CAD de diseño de referencia y modificarlos de acuerdo con las necesidades de cada uno. Para poder programar el microcontrolador en la placa Arduino se utiliza el lenguaje de programación basado en Wiring y el entorno de desarrollo basado en Processing.

Los proyectos realizados con Arduino pueden ejecutarse sin la necesidad de que estén conectados a un ordenador y además pueden comunicarse con distintos tipos de software como pueden ser Flash o Processing.

Arduino permite su instalación en Windows, Linux y Mac OS X. Podemos descargarnos la última versión del programa desde su página principal cuyo enlace es el siguiente:

http://arduino.cc/en/Main/Software

Una vez instalado el programa nos encontraremos con lo siguiente *(Figura 54)*:

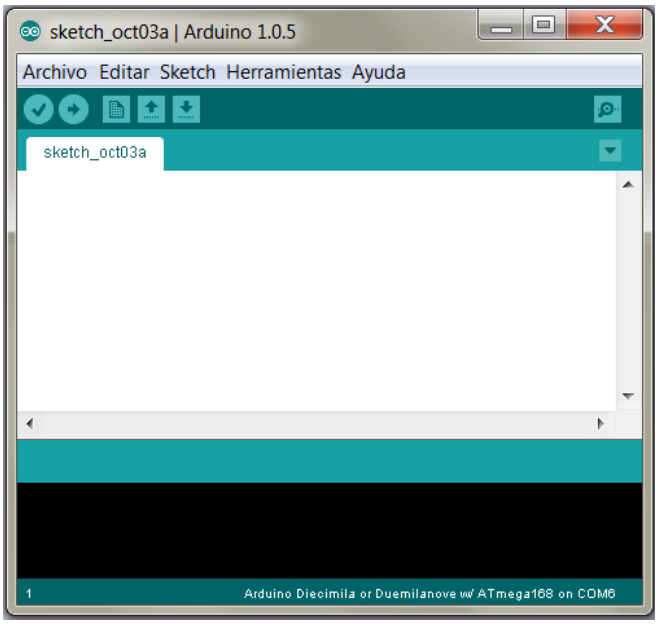


Figura 54: Entorno de Arduino

Es importante destacar que el programa se basa en la creación de Sketch. Un Sketch es el nombre que utiliza Arduino para escribir el software. Son los programas que posteriormente se cargan en la placa. Un Sketch puede estar compuesto por una o varias pestañas.

Tenemos un programa de aspecto sencillo en el que se distinguen cuatro partes distintas:

- Barra de herramientas. Desde aquí controlamos las opciones más importantes del programa, pudiendo entre otras cosas elegir el tipo de tarjeta que vayamos a usar, importar librerías o ver distintos ejemplos de sketch ya creados.
- Botones de acceso directo. Son acceso a las opciones más usadas a la hora de crear un programa. Con ellos podemos verificar el código escrito, abrir o guardar los avances. Aquí también tenemos acceso al botón de acceso directo del "Monitor Serial", con él podemos comunicarnos con el programa una vez esté cargado y funcionando en la placa y nos mostrará por pantalla la información en tiempo real de todo aquello relativo al proyecto y que haya sido programado previamente.
- Ventana principal. Corresponde a la zona central del programa, y es en esta parte donde se desarrolla todo el código del Sketch. Es nuestro editor de texto.
- Ventana inferior o de información. Es la que se encarga de mostrar los errores que tenga el código y de informar cuando se ha establecido contacto con la placa.

5.3. Programación

Una vez que hemos visto un poco como es el entorno de Arduino y cómo funciona, es el momento de explicar el programa que se ha creado para que el robot funcione y siga unos objetivos.

Antes de nada veamos cómo conectamos el robot al ordenador para poder transmitirle los datos *(Figura 55)*. Como hemos indicado anteriormente para este propósito se utiliza un adaptador foca de cinco pines. Este adaptador se conecta al ordenador mediante un puerto USB mini. Los cinco pines son los que se conectarán al robot tal y como indica la imagen.

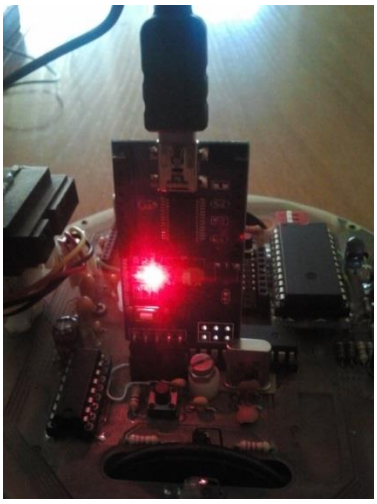


Figura 55: Conexión robot-ordenador

Utilizamos el botón de cargar para introducirle la programación. Para que los datos se transfieran correctamente es importante que el robot se encuentre conectado o el programa de Arduino dará error.

Otro factor que tenemos que tener en cuenta es la configuración del programa de acuerdo a nuestro caso. Para ello tenemos que irnos a la barra de herramientas y en la pestaña de herramientas elegir las siguientes opciones:

- Tarjeta → Arduino Diecimila or Duemilanove w/ ATmega168
- Puerto Serial → COM6 (esta opción se habilita cuando se encuentra conectado el USB mini y el adaptador foca).
- Programador → AVRISP mkII

Una vez tenemos todo listo con los parámetros necesarios para que podamos trabajar, comenzaremos a explicar el programa en función de los objetivos marcados.

Otro dato importante es la relación que existe entre los pines de conexionado y Arduino. Debemos saber que pin corresponde con cada elemento del robot para así poder utilizarlo. En la siguiente tabla podemos ver cuáles son.

Pin Arduino	Pin AVR	Notas	Uso
0	PD0	RX	Conectado al TX del FTDI, Bluetooth o XBee
1	PD1	TX	Conectado al RX del FTDI, Bluetooth o XBee
2	PD2	INT0	PUSH, SERVO
3	PD3	INT1, PWM	SpeedL
4	PD4		DIR_L
5	PD5	PWM	PWROUT, IRLEDs
6	PD6	PWM	PWROUT, SERVO
7	PD7		DIR_R
8	PB0		A
9	PB1	PWM	B
10	PB2	PWM	C
11	PB3	PWM, MOSI	SpeedR
12	PB4	MISO	D
13	PB5	SCK	ServoScanner
A0	PC0		MUX
A1	PC1		SERVO, R
A2	PC2		SERVO, G
A3	PC3		SERVO, B
A4	PC4	I2C_SDA	
A5	PC5	I2C_CLA	

Nota: Las salidas del PWM (modulación por ancho de pulso) creadas en los pines 5 y 6 tendrán ciclos de trabajo superiores a los esperados. Esto es debido a las interacciones con las funciones "*delay()*" y "*millis()*", que comparten el mismo temporizador interno utilizado para generar las salidas PWM. Esto se notará principalmente en la configuración de ciclo de servicio bajo (por ejemplo, 0 - 10) y puede resultar que en un valor de 0 no apague completamente la salida en los pines 5 y 6.

Nota: Los PWM de los pines 9 y 10 se deshabilitan por la librería Servo.h.

También es importante mostrar la tabla que relaciona las entradas del multiplexor con el sensor correspondiente. Elegiremos uno u otro en función de la necesidad y se mostrará por la salida del multiplexor (correspondiente con A0 en Arduino).

Pin MUX	Uso
0	
1	
2	
3	
4	
5	
6	
7	
8	encLA

9	encLB
10	encRA
11	encRB
12	US0
13	US1
14	IR0
15	$0.4*V_{in}$

En nuestro diseño, los pines están asociados a los encoders, los sensores de ultrasonidos y el sensor infrarrojos.

La estructura que seguiremos para explicar las funciones y objetivos será como la indicada en el apartado anterior; es decir, declararemos al principio del programa todas aquellas funciones y líneas de código necesarias para el funcionamiento básico del programa y a continuación explicaremos los programas y el código impuesto en "*setup*()" y "*loop*()". Llegado el momento lo explicaremos más detalladamente.

5.3.1. Identificación de pines

Lo primero que debemos hacer es identificar y asociar los pines del robot con el programa *(Figura 56)*. La relación de los pines y los elementos a los que se refieren podemos verla en el apartado anterior. Definimos cada uno de los pines: multiplexor (entrada y salida), leds, servo y motores.

```
// Multiplexer:
const int MUX_X_PIN = A0;

const int MUX_A_PIN = 8;
const int MUX_B_PIN = 9;
const int MUX_C_PIN = 10;
const int MUX_D_PIN = 12;

// H-Bridge:
const int L_DIR_PIN = 4;
const int L_VEL_PIN = 3;
const int R_DIR_PIN = 7;
const int R_VEL_PIN = 11;

// RGB LEDs:
const int LED_R_PIN = A1;
const int LED_G_PIN = A2;
const int LED_B_PIN = A3;

// IR LEDs:
const int LED_IR_PIN = 5;

// Test Pushbutton:
const int PUSH_PIN = 2;

// Servo Scanner:
const int SERVO_PIN = 13;
```

Figura 56: Identificación de pines

Las entradas del multiplexor corresponden con los pines 8, 9, 10 y 12 de Arduino. Para la salida corresponde A0.

Para los motores se usan los pines 3, 4, 7 y 11. Se necesitan dos por cada uno de ellos. El servo por su parte está asociado al pin 13.

El led RBG tiene como pines de Arduino a A1, A2 y A3. Para el led del sensor infrarrojo se utiliza el 5.

Se utilizará el pin número 2 para el botón de pulsación de reinicio del robot cuando éste esté encendido.

También debemos incluir las librerías del servo *(Figura 57)*, para que el robot lo reconozca y permita su control. Así mismo declaramos como variable global al servo.

```
#include <Servo.h>

Servo servoIR;
```

Figura 57: Librería servo

5.3.2. Definición de funciones

En esta sección nos encargaremos de mostrar y explicar las distintas funciones que componen el programa. Son de vital importancia pues al crearlas y hacer el llamamiento de alguna de ellas podremos configurar el robot según queramos.

También veremos que algunas de estas funciones incluyen en su código otras de las funciones indicadas anteriormente. Debido a que a medida que avanza el programa la programación se hace más compleja, será necesario en ocasiones llamar a alguna de ellas para formar parte de otra función aún mayor.

5.3.2.1. Funciones principales

5.3.2.1.1. Selección de salida del multiplexor

La función del multiplexor *(Figura 58)* es básica, puesto que es la que nos va a permitir posteriormente activar unos sensores u otros.

```
void setMux(int D, int C, int B, int A){

  digitalWrite(MUX_D_PIN , D);
  digitalWrite(MUX_C_PIN , C);
  digitalWrite(MUX_B_PIN , B);
  digitalWrite(MUX_A_PIN , A);
}
```

Figura 58: Función del multiplexor

El funcionamiento es simple. La función "*setMux*" tiene cuatro entradas: A, B, C y D; una por cada entrada del multiplexor con Arduino. El multiplexor por su parte tiene 16 pines y cada uno de ellos se asocia con un sensor distinto (no se utilizan todos, habrá más o menos en función de los sensores que le acoplemos).

Mediante el comando "*digitalWrite(,)*" llamamos a las distintas entradas definidas al principio y se les asociará uno de los valores según corresponda (A, B, C o D). Para activar uno de los 16 pines del multiplexor, deberemos escribir su valor en número binario. Por tanto las entradas solo pueden tener como valores 0 o 1.

La suma de todas las entradas nos dará como resultado el pin que queremos activar y que corresponde con uno de los sensores. Pongamos un ejemplo para verlo mejor.

Si realizamos este llamamiento de la función: "*setMux(1,1,0,0)*" le estamos indicando a Arduino que queremos activar el pin X12.

5.3.2.1.2. Movimiento de motores

Esta función es la que nos permite controlar y elegir la velocidad de movimiento de cada uno de los motores de continua *(Figura 59)*. Tiene como valores de entrada la velocidad deseada para cada uno de los motores. El rango de valores que puede tomar es de 255 hasta -255; siendo el 0 el valor mediante el cual el motor se encuentra parado y los valores negativos indican que el motor se mueve en la dirección contraria.

```
void setMotorVal(int lv, int rv){

  // Left motor
  if (lv>255) lv = 255;
  if (lv<-255) lv = -255;
  if (lv>=0){  // CW
    digitalWrite(L_DIR_PIN,LOW);
    analogWrite(L_VEL_PIN,lv);
  }
  if (lv<0){  // CCW
    digitalWrite(L_DIR_PIN,HIGH);
    analogWrite(L_VEL_PIN,255 + lv);
  }

  // Right motor
  if (rv>255) rv = 255;
  if (rv<-255) rv = -255;
  if (rv>0){  // CW
    digitalWrite(R_DIR_PIN,HIGH);
    analogWrite(R_VEL_PIN,255 - rv);
  }
  if (rv<=0){  // CCW
    digitalWrite(R_DIR_PIN,LOW);
    analogWrite(R_VEL_PIN, -rv);
  }
}
```

Figura 59: Función de velocidad de los motores

Veamos el motor izquierdo (la mecánica es la misma para el otro motor). Para poder realizar esta función hemos de tener en cuenta dos parámetros del robot: L_VEL_PIN y L_DIR_PIN. El primero indica la velocidad del motor; el segundo la dirección de giro.

En el caso de que la velocidad requerida sea positiva, el valor de L_DIR_PIN se mantendrá apagado (LOW) y la velocidad a la que girará será el valor que se haya dado (en el caso de que exceda de 255, tomará el valor máximo permitido dentro del rango).

Cuando el valor del motor sea negativo; L_DIR_PIN se mantendrá activo (HIGH) haciendo que el motor gire en dirección contraria al caso anterior. Del mismo modo la velocidad de giro será la que le hayamos indicado como valor de entrada.

La rueda derecha funciona igual; sin embargo se encuentra situada de forma opuesta a la rueda izquierda. Por eso las condiciones de velocidad y dirección son. Con esto conseguimos que ambas ruedas vayan siempre en la dirección que queramos.

Los movimientos serán rectilíneos si ambos motores van en la misma dirección, o bien serán giros si las ruedas se mueven en direcciones opuestas.

5.3.2.1.3. Sensor de ultrasonidos

Nos encontramos ante uno de los objetivos requeridos para el funcionamiento del robot. Nuestro robot consta de dos sensores de ultrasonidos situados uno a cada lado. Este tipo de sensores recibe también el nombre de "*PING)))*". La estructura de la función de ultrasonidos *(Figura 60)* es la siguiente.

```
int readUltrasonicSensor(char select){

  switch (select) {
    case 'L':
      setMux(1,1,0,0); // Left ultrasonic sensor (X12)
      break;
    case 'R':
      setMux(1,1,0,1); // Right ultrasonic sensor (X13)
      break;
    default:
      Serial.println("Error selecting ultrasonic sensor position");
    }

  long duration;

  pinMode(MUX_X_PIN, OUTPUT);
  digitalWrite(MUX_X_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(MUX_X_PIN, HIGH);
  delayMicroseconds(5);
  digitalWrite(MUX_X_PIN, LOW);

  pinMode(MUX_X_PIN, INPUT);
  duration = pulseIn(MUX_X_PIN, HIGH);

  return (duration/29)/2;
}
```

Figura 60: Función de sensores de ultrasonido

El funcionamiento de este tipo de sensores se basa en que el PING))) detecta la distancia que existe entre el sensor y el objeto más cercano a éste. El sensor envía una ráfaga de ultrasonido y espera a escuchar el eco producido por el objeto cuando choca contra él.

Arduino envía un pulso para enviar esta detección y espera a recibir un nuevo pulso por el mismo pin por el que lanza el primero. El tiempo que tarda entre que lanza el primer pulso y recibir el segundo será el que refleje la distancia que existe entre el sensor y el objeto.

Conocida esta distancia y sabiendo la velocidad a la que se desplaza el sonido, podemos calcular la distancia a la que se encuentra. Veamos la forma de reflejar este concepto en el programa.

En primer lugar debemos indicar como valor de entrada que sensor es el que queremos activar. En función de cual elijamos, se seleccionará la entrada del multiplexor correspondiente (X12 para el izquierdo o X13 para el derecho).

Una vez tenemos elegido el sensor que queremos que nos mida la distancia crearemos una variable que será la que recoja este valor.

Para asegurarnos de que enviamos un pulso limpio y así obtener un resultado fiable, producimos un pulso (HIGH) de dos microsegundos.

A continuación pasamos a recoger la señal del PING))). Para ello asociamos la variable a la salida del multiplexor y le pedimos que mande un pulso. El pulso estará activo hasta que la señal rebote contra el objeto y su eco vuelva de nuevo al sensor. De este modo obtenemos el tiempo en microsegundos que ha tardado en enviar y volver a recibir la señal.

Ahora lo único que queda por hacer es pasar este resultado a una distancia métrica. Para ello sabemos que la velocidad del sonido es de 340 m/s o lo que es lo mismo 29 centímetros por microsegundo. Dividimos el resultado entre la velocidad y obtenemos la distancia total.

Como la señal es de ida y vuelta, será necesario también dividir el resultado entre 2 para obtener así la distancia a la que se encuentra el objeto.

5.3.2.1.4. Sensor de infrarrojos

Es el turno de hablar del otro tipo de sensor que nos sirve para recoger datos referidos a la distancia y posicionamiento de los objetos.

El correcto funcionamiento del sensor infrarrojo es otro de los objetivos que debe cumplir el robot. El código creado para que esto sea posible es el siguiente *(Figura 61)*.

```
float DistanciaIR;

int readIRSensor(){

  setMux(1,1,1,0);  // X14
  int sensorIR = analogRead(MUX_X_PIN);

  float tension = sensorIR*0.0048828125;
  DistanciaIR = 27.041*pow(tension, -1.189);

  return DistanciaIR;
}
```

Figura 61: Función de sensor de infrarrojo

En primer lugar declararemos la variable que nos muestre el resultado final de este sensor. Para conseguir un resultado fiable hay que hacer unas cuantas operaciones.

Lo siguiente que debemos hacer es activar el pin correspondiente a este sensor y le asociamos una variable para que su valor pueda ser recogido. El valor que obtenemos está comprendido entre 0 y 1023. Deberemos cambiar este valor a uno relacionado con la tensión puesto que este resultado está referido al conversor de 10 bits.

El sensor se encuentra alimentado por una tensión de 5 V, así que deberemos dividir este valor de tensión máximo entre el total del número de bits que recoge el sensor.

$$valor\ del\ sensor * \frac{5\ V}{1024}$$

Una vez que tenemos el resultado en valores de tensión, lo siguiente que debemos hacer es convertirlo en distancia. Para ello necesitamos conocer la relación que existe entre tensión y distancia de reflexión del sensor. En la hoja de especificaciones podemos ver una gráfica con esta relación *(Figura 62)*.

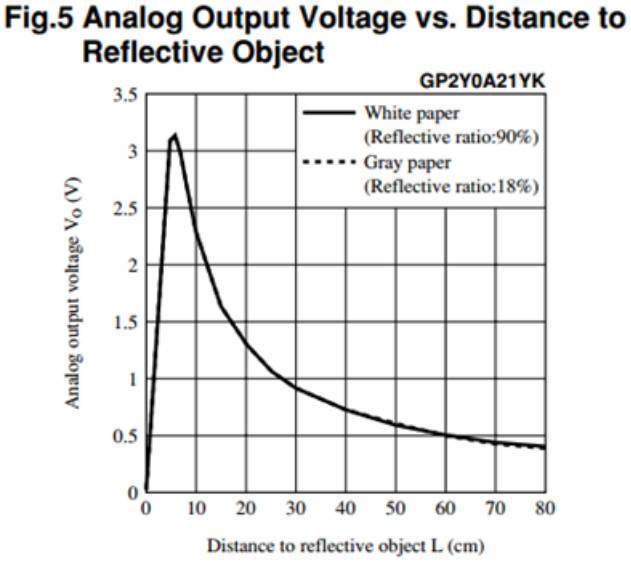


Figura 62: Gráfica tensión/distancia sensor de infrarrojo

La gráfica puede ser distinta en función del modelo de sensor que tengamos. En nuestro caso es el modelo con referencia GP2Y0A21YK0F.

Si miramos la gráfica podemos comprobar que no sigue una línea recta y en algunos casos toma dos valores. Por ejemplo para una tensión de 1 V le correspondería una distancia de 2 y 28 cm. La solución más sencilla correspondería con fijar un valor y aproximar la gráfica a una recta. Esta solución aunque posible no es muy precisa por lo que necesitamos un modo más fiable de obtener los valores.

Por tanto tenemos que buscar otra ecuación que sea más fiable que la recta. Si nos fijamos la curva desde el valor en el que empieza a decrecer sigue una función de tipo potencial. Cogiendo los puntos de la gráfica podemos trazar nuestra ecuación.

Así pues tomando valores a partir de 10 cm podemos saber qué valores tomaría nuestra función. Para ayudarnos con esto podemos utilizar cualquier programa que nos permita calcular gráficas y funciones (ej. Excel).

Los datos y la gráfica resultante son los siguientes *(Figura 63)*.

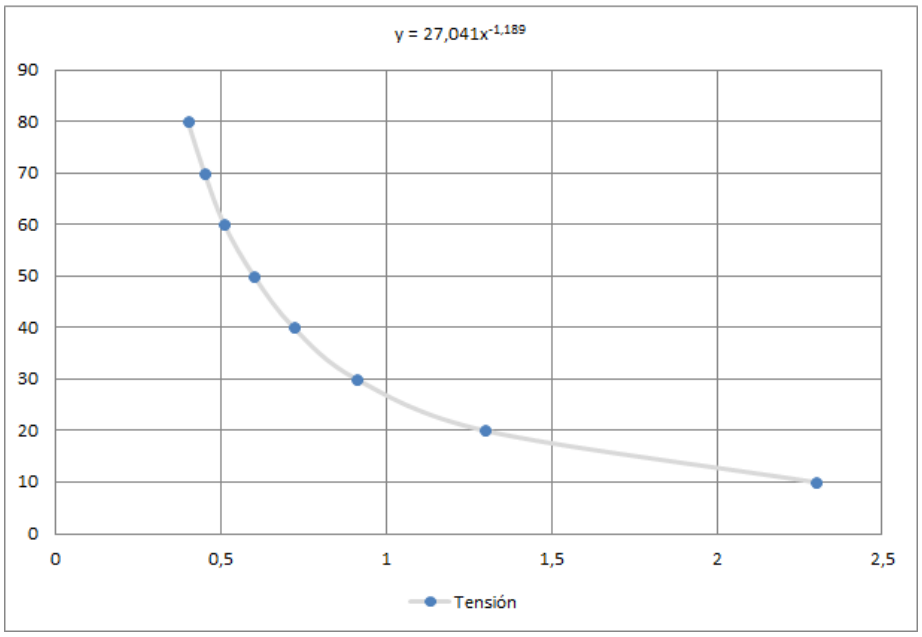


Figura 63: Gráfica distancia/tensión sensor de infrarrojo

Podemos observar que la gráfica está representada habiendo intercambiado la posición de los datos en los ejes X e Y con respecto a la gráfica anterior de especificaciones. Así obtenemos el valor de la distancia a partir de la tensión.

Los parámetros de esta nueva función serán los que incluyamos en el programa. Utilizando la función *"pow()"* con esos datos podemos obtener la distancia.

$$y = 27.041\ x^{-1.189}$$

Este tipo de sensores tienen como inconveniente que no son capaces de medir objetos transparentes; del mismo modo podemos obtener un valor erróneo si sobre el sensor actúa directamente una fuente de luz que no sea la del propio sensor.

Esta función funciona bastante bien por sí sola; no obstante en ocasiones podemos obtener un resultado erróneo que se sale de lo esperado. Esto puede ocurrir por diversos motivos como pueden ser efectos de luz que incidan en el sensor o bien que éste incida sobre superficies reflexivas.

Es importante no encontrarnos con alguno de estos valores falsos puesto que el programa del robot interpretaría esos resultados dando como resultado una acción equivocada.

Para evitar esto se ha diseñado otra función que nos dé una mayor precisión sobre la distancia detectada. La función es la siguiente *(Figura 64)*.

```
int medidaIRSensor(){
  int ir[2]; int valorIR;
  for (int i=0; i<3; i++){
    ir[i] = readIRSensor();
    delay(75);
  }
  if (abs(ir[0] - ir[1]) < 2){
    valorIR = ir[0];
  }
  else{
    if (abs(ir[0] - ir[2]) <= abs(ir[1] - ir[2])){
      valorIR = ir[0];
    }
    else{
      valorIR = ir[1];
    }
  }
  return valorIR;
}
```

Figura 64: Función de sensor de infrarrojo 2

Utilizando la función anterior realizamos un total de tres medidas distintas que son guardadas en una pequeña matriz. A continuación la función compara los primero valores obtenidos para comprobar si tienen valores similares; de ser así la función terminaría dando como resultado final la primera de las medidas tomadas.

En el caso de que los valores difieran mucho entre sí pasaremos a comparar estos dos valores con la tercera medida. Mirando en que caso los valores son más próximos con respecto a este último, elegiremos la primera o segunda medida realizada.

5.3.2.1.5. Movimiento del servo

Lo siguiente que debemos hacer es realizar el movimiento del servo. Al principio del programa habíamos incluido la librería e identificado el servo. Ahora mediante la siguiente función podemos moverlo *(Figura 65)*.

```
int anguloSv;

int Mservo(int IR){
  for (anguloSv = 0; anguloSv < 180; anguloSv ++){
    servoIR.write(anguloSv);
    if (IR=1){
      readIRSensor();
      Serial.print("Infrarrojo [cm]: "); Serial.println(DistanciaIR);
    }
    delay(100);
  }

  for (anguloSv = 180; anguloSv >= 1; anguloSv --){
    servoIR.write(anguloSv);
  }
  while (anguloSv = 1){
    break;
  }
}
```

Figura 65: Función del servo

El servo se encuentra situado en la parte delantera del robot, en la parte central. Puede programarse de diversas maneras en función de lo que queramos que haga. En nuestro caso el objetivo que queríamos alcanzar con él es que pudiera moverse de tal manera que el sensor de infrarrojo situado encima fuese capaz de detectar los objetos en un radio de 180 º.

Es por eso por lo que el servo está programado para que realice un barrido de giro de esa extensión. Cuando llega al final de su recorrido, vuelve a su posición original.

Del mismo modo y, al estar tan relacionado con el sensor IR, se ha creado la opción de que al llamar a la función podamos al mismo tiempo hacer una lectura con el sensor. Para realizar esto, es necesario incluir como valor de entrada de llamada a la función un 1, para indicar que se activa también el sensor.

Tras realizar esta primera versión de la función que controla el servo, hemos realizado una nueva más enfocada a detectar la distancia más pequeña en un cierto rango de amplitud. Esta función tiene el mismo nombre que la anterior, pero ahora además podemos regular el rango del servo así como el incremento del ángulo. Esta función la veremos más adelante, cuando nos centremos en las funciones del programa principal.

5.3.2.1.6. Función de los encoders

Como hemos dicho anteriormente, el robot consta de un total de cuatro encoders, dos por cada rueda. Los encoders serán los encargados de controlar la dirección y el movimiento de los motores.

En primer lugar tendremos que crear la función que llame a cada uno de los encoders *(Figura 66)*.

```
int readEncoder1(){
  setMux(1,0,0,0);
  return(analogRead(MUX_X_PIN));
}

int readEncoder2(){
  setMux(1,0,0,1);
  return(analogRead(MUX_X_PIN));
}

int readEncoder3(){
  setMux(1,0,1,0);
  return(analogRead(MUX_X_PIN));
}

int readEncoder4(){
  setMux(1,0,1,1);
  return(analogRead(MUX_X_PIN));
}
```

Figura 66: Función de los encoders

Estas funciones son básicas y servirán más adelante para poder controlar las ruedas.

El funcionamiento de estos sensores consiste en captar la luz que les llega por un led situado en frente de ellos. En función de la luz que capten darán como resultado un valor numérico de más o menos valor. El rango está comprendido entre 0 y 1200.

El sensor captará más luz cuanto menor sea el valor del rango. En caso de no captarla este valor será mayor.

5.3.2.1.7. Función de los contadores

Para poder controlar los motores y su velocidad debemos crear antes un contador que cuente en cada momento los pequeños incrementos de movimiento de las ruedas.

Para cumplir con este objetivo es muy importante el diseño de las ruedas, ya que las cavidades de su diseño son las encargadas de decir al encoder cuanto se han movido.

Estos agujeros se encuentran en dos alturas distintas y desfasados un 50%. La altura a la que se encuentran las cavidades está pensada para que coincidan con los encoders (uno para la cavidad superior y otro para la inferior).

Cuando se conectan los motores y a medida que las ruedas van avanzando en su recorrido, los encoders irán captando la luz que les llega de los leds situados enfrente cuando pase por su posición uno de los huecos. Cuando coincida el encoder con una parte sólida de la rueda, no será capaz de captar la luz.

De este modo conseguimos una secuencia de luz/sombra que nos permitirá llevar la cuenta de las veces que la rueda ha sufrido un incremento de movimiento. La secuencia que sigue es la siguiente *(Figura 67)*.

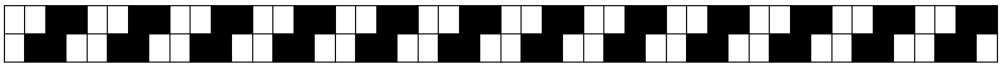


Figura 67: Secuencia luz/sombra de las ruedas

La rueda al girar hace que en cada momento los encoders vean una de esas posiciones. El cuadro en blanco representa que le llega la luz; el negro que no le llega. Mirando la imagen y considerando las columnas, existen un total de cuatro posiciones distintas.

```
int Encoder1B = 0; int Encoder1C = 0; int Encoder2B = 0; int Encoder2C = 0;
int contadorL1 = 0; int contadorL3 = 0; int contadorL2 = 0; int contadorL4 = 0;
int contadorL  = 0;

int contadorI(){                        //Contador Izquierdo//
  if (contadorL >= 0) {
    int Encoder1A = readEncoder1();               //Encoder Izquierdo Superior [D7]
    if (Encoder1A > 750){
      Encoder1B = Encoder1A;
    }
    if (Encoder1A < 250){
      Encoder1C = Encoder1A;
    }

    if (Encoder1A < 250){
      if (Encoder1A < Encoder1B){
        contadorL1 = contadorL1 + 1;
      }
      Encoder1B = 0;
    }
    if (Encoder1A > 750){
      if (Encoder1A > Encoder1C){
        contadorL3 = contadorL3 + 1;
      }
      Encoder1C = 1200;
    }

    int Encoder2A = readEncoder2();                 //Encoder Izquierdo Inferior [D4]
    if (Encoder2A > 750){
      Encoder2B = Encoder2A;
    }
    if (Encoder2A < 250){
      Encoder2C = Encoder2A;
    }
    if (Encoder2A < 250){
      if (Encoder2A < Encoder2B){
        contadorL2 = contadorL2 + 1;
      }
      Encoder2B = 0;
    }
    if (Encoder2A > 750){
      if (Encoder2A > Encoder2C){
        contadorL4 = contadorL4 + 1;
      }
      Encoder2C = 1200;
    }
    contadorL = contadorL1 + contadorL2 + contadorL3 + contadorL4;
    return contadorL;
  }
}
```

Figura 68: Función de los contadores

El objetivo de nuestra función *(Figura 68)* es que sea capaz de reconocer cada uno de esos estados y los incluya en un contador que indique cuantos incrementos han pasado desde que los motores se han puesto en movimiento.

Esta función se divide en dos secciones fundamentales, una para el encoder superior y otra para el inferior. En ambas la mecánica es la misma. Veámoslo por partes.

En primer lugar será necesario declarar las variables que vamos a usar durante la función. Serán necesarias declarar unas variables que nos permitan guardar cada uno de esos estados, así como otra que guarde la suma total de las anteriores.

También serán necesarios crear unas variables que nos guarden el estado previo al que estamos comparando para así saber si se ha pasado a un nuevo estado o si por el contrario seguimos en el mismo.

A continuación llamamos a la función que activa el encoder y creamos dos opciones: una para el caso de que detecte luz (el valor obtenido tiene que ser menor a 250) y otra para cuando no la detecte (el valor obtenido tiene que ser mayor de 750). En ambos casos le pedimos a la función que guarde el valor para una posterior comparación.

Hemos establecido los límites en 250 y 750 porque a mayores velocidades es más difícil para el encoder detectar los huecos y necesita más margen para captarlos.

Después le pedimos a la función que mire el valor actual y lo compare con el valor previo que se ha guardado en la iteración anterior. Aquí también tenemos dos casos:

- Si cumple que el estado actual es menor de 250 y el estado anterior es mayor de 750, significa que se ha producido un cambio y que tenemos una iteración, por lo que añadimos 1 a un contador parcial para cada vez que tengamos esta situación.
- Es el mismo caso que el anterior salvo que aquí mira que el estado actual sea mayor de 750 y el estado anterior menor de 250.

Tras añadir 1 a cualquiera de los contadores parciales, debemos garantizar que en la siguiente iteración que aún podría estar en la misma franja no vuelva a contarla por error. Por eso establecemos que el estado previo tome un valor máximo o mínimo que sea imposible de alcanzar por el encoder. Solo tras una nueva iteración que cumpla las condiciones impuestas al principio de la función volveremos a guardar nuevos valores.

Lo que tratamos de buscar con estas comparaciones es el paso de luz a sombra que se produce en los bordes de los huecos, de este modo garantizamos que nunca cuente una iteración de más.

De este modo evitamos además aquellos estados comprendidos entre 250 y 750, ya que en ellos aún no se ha producido el cambio de iteración.

Este mismo proceso se realiza con el encoder inferior. De este modo tenemos un total de cuatro contadores parciales que se van incrementando a medida que pasamos de una iteración a otra.

Tan solo quedará sumar todos los contadores parciales para llevar la cuenta total del número de iteraciones realizadas por el motor en un tiempo determinado.

Debemos mencionar que esta misma función se realiza para el otro motor; la única diferencia vendrá dada por los encoders que utiliza.

5.3.2.1.8. Función de velocidad

Una vez tenemos la forma de contar el número de incrementos que sufre la rueda en su desplazamiento, es el momento de convertir estos incrementos en una velocidad que podamos medir y controlar.

La mecánica para conseguir esto se basa en el número de incrementos que se producen en un giro de vuelta.

Como hemos visto en el apartado anterior, nuestras ruedas siguen una secuencia de luz/sombra que es la que nos permite contar el movimiento. En este caso las ruedas están diseñadas de tal forma que tenemos un total de 48 incrementos o pulsos por rueda.

Esto significa que cuando el contador haya llegado a 48, la rueda habrá dado una vuelta completa. Esta relación será la que nos permita relacionar los pulsos con la velocidad, que expresaremos en rpm.

Para calcular la velocidad en rpm utilizaremos la siguiente expresión:

$$vel\ [rpm] = \frac{\frac{\Delta Encoder}{\Delta Tiempo} * \frac{60\ [s]}{1\ [min]} * \frac{1000\ [ms]}{1\ [s]}}{48\ [pulsos/vuelta]}$$

La velocidad se calcula midiendo el número de pulsos que captan los encoders en un tiempo determinado. Como el programa mide el tiempo en milisegundos, es necesario pasar esa unidad a minutos. Del mismo modo 48 expresa el número máximo de pulsos que puede producirse en una revolución o vuelta.

En el caso de diseñar otras ruedas con un número distinto de cavidades a las que tenemos actualmente, será necesario modificar el valor de pulsos/vuelta.

Veamos cómo quedaría esta idea en el programa de Arduino *(Figura 69)*.

```
int contadorLprev = 0; double velMotorL = 0;          //Motor Izquierdo//
unsigned long tiempo2 = 0;
int mediaL = 1; double acumulacionL = 0;

int velMotorI(){
  unsigned long tiempo1 = millis();
    if(tiempo1 >= tiempo2 + 250){
      velMotorL = (1250*(double)(contadorL-contadorLprev))/(double)(tiempo1-tiempo2);
      contadorLprev = contadorL;
      tiempo2 = tiempo1;
      if (mediaL > 5){
        acumulacionL = acumulacionL + velMotorL;
        velMotorL = acumulacionL/(mediaL - 5);
      }
      mediaL++;
    }
}
```

Figura 69: Función de la velocidad

En primer lugar y, como en casos anteriores, el primer punto consiste en definir las variables que vamos a necesitar para desarrollar este programa.

En esta ocasión necesitamos una variable que recoja la velocidad a mostrar. Del mismo modo también serán necesarias unas variables de tiempo para poder tener un control del intervalo que se esté analizando, así como un contador que nos permita saber la cantidad de pulsos o incrementos que se han producido en el intervalo anterior.

El programa funciona calculando en un determinado periodo de tiempo el número de pulsos que han sido capaces los encoders de captar. Esto se produce de forma repetitiva en todas las iteraciones mientras los motores se encuentren en movimiento.

Una vez declaradas las variables comenzamos a analizar la función. Al principio establecemos la frecuencia de tiempo con la que llamaremos a la función. Este tiempo será de 250 milisegundos. De este modo tenemos un tiempo constante que nos permitirá calcular el incremento o decremento de pulsos contados en cada iteración.

A continuación definimos la función de velocidad indicada anteriormente. Para simplificar los datos, hemos puesto directamente el producto de multiplicar todas las constantes de la ecuación.

$$1250 = \frac{60 * 1000}{48}$$

Los valores del contador y del tiempo se guardan para la siguiente iteración.

Debido a la limitación de huecos en nuestras ruedas vemos que aunque la velocidad de los motores sea constante, los contadores no cuentan siempre el mismo número de pulsos. En ocasiones los encoders ven un pulso más o un pulso menos. Esto hace que el resultado de la velocidad expresado por la ecuación fluctúe entre distintos valores. Si las ruedas tuvieran un mayor número de cavidades este error sería menor.

En nuestro caso la solución para conseguir un valor estable que corrija esta diferencia de velocidades entre iteraciones consiste en aplicarle una media al resultado de velocidad obtenido. De este modo cuanto más avance el robot a una cierta velocidad, más estable se volverá su valor. Si tenemos variaciones continuas de velocidad, esta solución no será factible y será mejor prescindir de ella.

Esta función se realiza también para la otra rueda. Habrá que tener en cuenta por tanto el contador perteneciente a esa rueda.

5.3.2.1.9. Funciones de movimiento

Una vez establecida la forma de representar la velocidad a la que se mueve nuestro robot, ha llegado la hora de crear las funciones necesarias para su desplazamiento.

En este apartado vamos a ver tres funciones distintas: una de desplazamiento a un punto concreto y otras dos (avance y retroceso) de desplazamiento indefinido a distintas velocidades. La primera de estas ecuaciones estará incluida también en las otras.

Veamos en primer lugar la función de desplazamiento a un punto concreto *(Figura 70)*.

```
int avanceDistancia(int distancia, int RIzda, int RDcha){
  int medidor;
  int objetivo = 48*distancia/(2*PI*2.1);
  for (medidor = contadorL; objetivo >= contadorL - medidor;){
    contadorI();
    setMotorVal(RIzda,RDcha);
  }
  setMotorVal(0,0);
}
```

Figura 70: Función de desplazamiento definido

Esta función consta de tres variables de entrada:

- "*distancia*": este valor refleja la distancia que queremos que se desplace el robot. Este valor vendrá reflejado en centímetros.
- "*RIzda"*: indica el valor que debe tener el motor izquierdo.
- "*RDcha*": igual que el caso anterior pero esta vez referido al motor derecho.

Al incluir como valor de entrada una distancia en centímetros, el robot va avanzando hasta que realiza todo el recorrido, momento en el que se detiene. Para cumplir con este objetivo deberemos transformar el giro en movimiento lineal.

En primer lugar tendremos que saber lo que se desplazan las ruedas cada vez que éstas giran y dan una vuelta completa. Si el radio total de las ruedas es de 2.1 cm, bastará con hallar el perímetro para saber cuánto recorre en una vuelta.

$$2 * \pi * 2.1\ [cm] = 13.1946\ [cm/vuelta]$$

Una vez que sabemos que distancia es la que recorre el robot cuando las ruedas dan una vuelta, debemos averiguar cuanto avanza por pulso del encoder.

$$\frac{13.1946\ [cm/vuelta]}{48} = 0.2748\ [cm]$$

Para aplicarlo a nuestro programa deberemos averiguar cuantos pulsos son necesarios para alcanzar nuestro objetivo; para ello bastará con dividir a la distancia de entrada a la que queremos desplazarnos por lo que avanza en cada pulso. Establecemos un comparador y cuando alcanzamos el número de pulsos necesarios el robot se detiene.

Para el movimiento rectilíneo del robot hemos diseñado dos programas semejantes: uno para el avance y otro para el retroceso. El programa es similar en ambos, por lo que nos centraremos en uno de ellos para explicarlo.

El programa se ha diseñado de tal forma que consta de un total de cuatro velocidades distintas y dos modos de funcionamiento (modo continuo y distancia a un punto). Veamos por tanto el código del programa *(Figura 71)*.

```
double avanTotal;

int avance (int modo, int distancia){
  contadorI();
  contadorD();
  switch (modo) {
    case 1:
      setMotorVal(64,51);
      if (distancia > 0){
        avanceDistancia(distancia,64,51);
      }
      break;
    case 2:
      setMotorVal(127,111);//108
      if (distancia > 0){
        avanceDistancia(distancia,127,111);
      }
      break;
    case 3:
      setMotorVal(191,177);
      if (distancia > 0){
        avanceDistancia(distancia,191,177);
      }
      break;
    case 4:
      setMotorVal(255,245);
      if (distancia > 0){
        avanceDistancia(distancia,255,245);
      break;
      }
    default:
          Serial.println("Velocidad erronea");
  }
  avanTotal = contadorL*0.2748;
}
```

Figura 71: Función de movimiento

La función consta de dos variables de entrada:

- "*modo*": que establece la velocidad de desplazamiento
- "*objetivo*": para indicar la distancia a la que queremos desplazar el robot (si el valor es 0 el movimiento se produce ininterrumpidamente de forma continua).

Se han establecido cuatro modos de velocidad que corresponderían con un 25%, 50%, 75% y 100% de la potencia máxima que alcanzan los motores. En función de la situación requerida se podrá elegir la velocidad de acuerdo con nuestras necesidades.

En este caso la función elige un modo u otro en función de la velocidad deseada. Si queremos que el robot avance de forma indefinida el valor de entrada de la variable distancia debe ser 0; en caso contrario se introduciría cualquier otro valor, con lo que el programa llamaría a la función "*avanceDistancia()*" que hemos explicado anteriormente y así recorrería una distancia determinada.

5.3.2.1.10. Función de giro

La última función que permite el movimiento del robot corresponde a la del giro. Del mismo modo que el robot avanza, será necesario que gire en una dirección u otra en función de los obstáculos que pueda encontrarse o del recorrido a seguir.

Lo primero que tenemos que tener en cuenta para esta función es la relación que existe entre el giro de la rueda y la distancia desplazada durante ese giro *(Figura 72)*.

Por eso es importante conocer tanto el perímetro de la rueda como el perímetro de giro del propio robot.

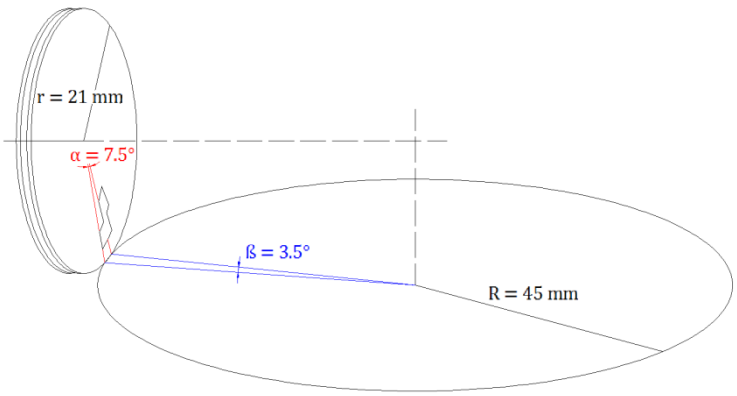


Figura 72: Relación de giro rueda/robot

En la imagen podemos apreciar esta relación. Nuestro objeto consiste en averiguar cuantos grados gira el robot por cada uno de los contadores de la rueda.

A la hora de realizar los cálculos denominaremos a las variables de la rueda con letras minúsculas y a las del robot con letras mayúsculas.

El primer paso para calcular los grados de giro por contador consiste en averiguar el perímetro de la rueda y del robot. El robot al ser circular y girar sobre sí mismo describirá una circunferencia cuyo diámetro es la distancia entre las ruedas.

$$p = 2 * \pi * r \ \rightarrow \ 2 * \pi * 21 \ \rightarrow \ p = 131.46 \ [mm]$$
$$P = 2 * \pi * R \ \rightarrow \ 2 * \pi * 45 \ \rightarrow \ P = 282.74 \ [mm]$$

Lo siguiente que debemos averiguar es la distancia que avanza la rueda a través de la circunferencia descrita por el giro del robot por cada pulso o contador de la rueda.

Primero calculamos el avance por contador de la rueda y posteriormente lo relacionamos con la otra circunferencia para saber cuántos pulsos le corresponderían.

$$\frac{131.46}{48} = 2.348 \ [mm/pulso]$$
$$C = \frac{282.74}{2.348} \rightarrow C = 102.85 \ pulsos$$

Solo queda por calcular el número de grados que le correspondería a la rueda cada uno de los pulsos de la circunferencia del robot.

$$\frac{360°}{102.85} \ \rightarrow \ \beta = 3.5°/pulso$$

Ese será el valor a incluir en la función para poder calcular correctamente el ángulo de giro del robot; debemos saber cuántos grados gira el robot por cada incremento. Veamos cómo quedaría la función correspondiente al giro *(Figura 73)*.

```
int giroRueda;

int giro (int grados, char orientacion){

  giroRueda = (grados/3.5)+0.5;

  setMotorVal(0,0);
  int referL;

  for (referL = contadorL; contadorL < referL+giroRueda-1;){
     contadorI();
     switch (orientacion) {
      case 'L':
        setMotorVal(-52,50);
        break;
      case 'R':
        setMotorVal(64,-50);
        break;
      default:
        Serial.println("Orientación erronea");
      }
  }
  setMotorVal(0,0);
  Serial.print("Giro [grados]: "); Serial.println(double(giroRueda+referL-contadorL)*3.5);
}
```

Figura 73: Función de giro

La función consta de dos variables de entrada: grados y orientación. La primera sirve para indicar el número de grados que queremos que gire el robot; la segunda es para indicar hacia donde debe girar el robot (izquierda o derecha).

Para que el robot gire los grados correctos, lo primero que debemos hacer es dividir el ángulo introducido entre 3.5°/pulso y así conocer el número de contadores.

Para aproximar lo máximo posible el ángulo pedido al movimiento del robot es necesario que sumemos 0.5 al ángulo dividido. De este modo si al dividir el ángulo entre 3.5 nos quedase un número decimal mayor de 0.5, al sumarle otra media décima el contador haría un pulso más. En caso de que al dividir quedase un número decimal menor de 0.5 aunque le sumásemos otra media décima no pasaría nada y el contador se quedaría con el valor que tuviese tras la división. Veamos algunos ejemplos:

- Con 70° obtenemos 20 pulsos exactos con un error de 0°.
- Si el ángulo fuese de 45° la división daría como valor final 13.35, por lo que haría un total de 13 pulsos. Si multiplicamos estos pulsos por 3.5 obtendremos 45.5°. De este modo el error que obtenemos es de 0.5°.
- Si tenemos 180° dividimos y tendríamos 51.42 pulsos. Al sumarle la media décima seguiríamos teniendo 51.92; por tanto haría 51 pulsos. En este caso al volver a multiplicarlo por 3.5 obtendríamos 178.5°, siendo el error de 1.5°.

Finalmente que nos queda contar los pulsos que se producen y compararlos con nuestro objetivo. Cuando este valor es alcanzado, el robot se detiene. En función de la orientación elegida, el robot girará a un lado u otro contando el número de pulsos.

5.3.2.1.11. Función de datos

Esta función *(Figura 74)* tiene como único objetivo el de informar por pantalla sobre todos aquellos datos que puedan ser relevantes sobre el robot.

```
int datos (){
  Serial.print("Ultrasonidos Izdo [cm]: "); Serial.println(readUltrasonicSensor('L'));
  Serial.print("Ultrasonidos Dcho [cm]: "); Serial.println(readUltrasonicSensor('R'));
  Serial.print("Infrarrojo [cm]: "); Serial.println(DistanciaIR);

  Serial.print("Contador Izdo: "); Serial.println(contadorL);
  Serial.print("Contador Dcho: "); Serial.println(contadorR);

  Serial.print("Velocidad Izdo (rpm): "); Serial.println(velMotorL);
  Serial.print("Velocidad Dcho (rpm): "); Serial.println(velMotorR);

  Serial.print("Distancia avanzada [cm]: "); Serial.println(avanTotal);
  Serial.print("Distancia retrocedida [cm]: "); Serial.println(retrTotal);
  Serial.print("Distancia total [cm]: "); Serial.println(avanTotal + retrTotal);

  Serial.println();
}
```

Figura 74: Función de datos

Muestra la información referida a los sensores, contadores, velocidad y distancia.

5.3.2.1.12. Función de parpadeo

En este caso nos encontramos ante una función destinada a encender y apagar el led RBG del robot. En función de los datos que le introduzcamos, el parpadeo que se produzca será de un color u otro. Veamos en que consiste la función *(Figura 75)*.

```
int parpadeo (int red, int green, int blue){
  digitalWrite(LED_R_PIN , red);
  digitalWrite(LED_G_PIN , green);
  digitalWrite(LED_B_PIN , blue);
  delay(500);
  digitalWrite(LED_R_PIN , LOW);
  digitalWrite(LED_G_PIN , LOW);
  digitalWrite(LED_B_PIN , LOW);
}
```

Figura 75: Función de parpadeo

Como se puede observar en la imagen, la función consta de tres variables de entrada, una por cada color primario de la luz (rojo, azul, verde). Si el valor de cualquiera de esos datos de entrada es cero, el color se encuentra desconectado; si el valor es uno significa que ese color interviene.

La combinación de los distintos valores dará lugar a un color u otro. En la siguiente tabla podemos comprobar que color corresponde con cada una de las posibilidades.

Combinación			Color Resultante
Rojo	Verde	Azul	
1	0	0	Rojo
0	1	0	Verde
0	0	1	Azul
1	1	0	Amarillo
1	0	1	Magenta
0	1	1	Cian
1	1	1	Blanco

Según la situación usaremos el color que mejor nos convenga.

5.3.2.2. Funciones de escaneo de habitación

En este apartado veremos todas aquellas funciones más involucradas directamente con la creación del programa principal del robot.

Dejando al robot en un punto de una habitación, éste debe ser capaz de realizar un escaneo inicial de 360° para ser capaz de encontrar la pared más cercana y dirigirse a ella para posteriormente recorrerla paralelamente hasta completar la habitación.

Así mismo el robot debe ser capaz de crear unos mapas de la zona en la que se encuentra; estos mapas se irán generando a medida que el robot se mueve.

5.3.2.2.1. Movimiento del servo

La primera función que encontramos en este apartado corresponde (como indicamos anteriormente) a una versión más completa y enfocada a este programa principal que la que vimos anteriormente. Veamos en que consiste *(Figura 76)*.

```
float minDistanciaIR=99999; int minAnguloSv; byte barrido=0; byte barridoprev=1;

int Mservo(byte IR, byte inicio, byte final, int incremento){
  int AnguloSv;
  servoIR.write(inicio);
  int DIR;
  delay(500);
  for (AnguloSv = inicio; AnguloSv <= final; AnguloSv = AnguloSv + incremento){
    servoIR.write(AnguloSv);
    if (IR=1){
      DIR = medidaIRSensor();
      Serial.print("Infrarrojo [cm]: "); Serial.println(DIR);

      if (DIR < minDistanciaIR){
        minDistanciaIR = DIR;
        minAnguloSv = AnguloSv;
        if (barrido != barridoprev){
          barrido = barridoprev;
        }
      }
    }
    delay(100);
  }
  barridoprev++;
  servoIR.write(90);
  Serial.print("Distancia minima [cm]: "); Serial.println(minDistanciaIR);
  Serial.print("Angulo [grados]: "); Serial.println(minAnguloSv);
  Serial.print("Barrido: "); Serial.println(barrido);
}
```

Figura 76: Función del servo 2

En este caso tenemos cuatro variables globales que serán usadas posteriormente por otras funciones. Todas estas variables corresponden con:

- *"minDistanciaIR"*: La mínima distancia detectada por el sensor de infrarrojos (al que inicializamos con un valor muy elevado).
- *"minAnguloSv"*: El ángulo en el cual ha encontrado ese valor mínimo.
- *"barrido"*: El barrido en el que se ha encontrado (en el caso de que la función se use en más de una ocasión, para así poder saber exactamente cuándo se ha encontrado el valor mínimo).
- *"barridoprev"*: El barrido previo (variable necesaria para llevar la cuenta del barrido en el que nos encontramos).

Como variables de entrada tenemos en este caso:

- "*IR*": si el valor es 1 indica que el sensor de infrarrojos está conectado.
- "*inicio*": valor que indica el ángulo inicial al que debe posicionar el servo (el mínimo es 0°).
- "*final*": indica el valor máximo que alcanza el ángulo del servo (máximo 180°).
- "*incremento*": es la amplitud de grados entre 2 ángulos consecutivos.

El funcionamiento de esta función se basa en ir recorriendo con el servo el rango de grados hasta completar el valor máximo de amplitud solicitado.

La función toma las medidas de la distancias recogidas por el sensor de infrarrojos. Este valor es almacenado hasta que toma un nuevo valor, momento en el que compara ambas medidas y guarda el resultado de la más pequeña de las dos. Este proceso lo realiza con cada incremento hasta terminar todos los grados.

Al final la función guarda como resultado el valor de la distancia mínima, su ángulo y en que barrido se ha realizado (en el caso de llamar a la función más de una vez).

5.3.2.2.2. Giro inicial

Una vez tenemos lista la función del servo, es el momento de ver la función *(Figura 77)* que será usada para hacer un escaneado inicial de 360° de toda la estancia.

```
int giroIn;

int escaneoInicial ( byte barridos, int escaneo){
  int gradoGI1 = 103/barridos;
  int gradoGI2 = gradoGI1 + 1;

  int g1 = (103-barridos*gradoGI2)/(gradoGI1-gradoGI2);
  int g2 = barridos - g1;

  int amplitud = 360/barridos;
  int amp1 = (180-amplitud)/2;
  int amp2 = amplitud + amp1;

  for (int i=0; i<g1; i++){
    Mservo(1,amp1,amp2,escaneo);
    giro(gradoGI1*3.5,'L');
  }
  int giroIn1 = gradoGI1*(barrido-1);

  for (int j=0; j<g2; j++){
    Mservo(1,amp1,amp2,escaneo);
    giro(gradoGI2*3.5,'L');
  }

  int giroIn2 = ((barrido-1)-g1)*gradoGI2;
  if (barrido>g1){
    giroIn = g1*gradoGI1 + giroIn2;
  }
  else{
    giroIn = giroIn1;
  }
  Serial.print("Giro: "); Serial.println(giroIn);
}
```

Figura 77: Función de giro inicial

En este caso el objetivo es que basándonos en la función anterior el robot debe escanear una zona de la habitación en busca del valor mínimo. Debido a que el servo está limitado a una amplitud máxima de 180°, es necesario que al menos el robot gire sobre sí mismo y escanee la zona que le queda para cubrir todo el espacio.

La función consta de dos variables de entrada:

- "*barridos*": este valor indica cuantos barridos queremos que realice el robot hasta completar una vuelta completa.
- "*escaneo*": nos indica la amplitud que debe haber entre dos medidas de un mismo barrido. Este valor se corresponde con la variable incremento de la función "*Mservo()*".

En primer lugar la función se encarga de establecer la amplitud de cada uno de los barridos. Para ello se basa en el número de contadores que debe tener cada uno. Sabemos que en nuestro caso una vuelta completa equivale a 102.85 contadores (puesto que el ángulo mínimo de giro es de 3.5° como ya vimos en la función de giro) que aproximaremos a un total de 103 contadores.

Puesto que es un número irregular, los barridos siempre serán distintos. La función se centra en realizar los barridos de dos tamaños distintos: *x* y *x+1* (corresponden con las variables "*gradoGI1*" y "*gradoGI2*"). En función del número de barridos que queramos hacer habrá más de uno u otro.

Lo siguiente es por tanto saber el número de barridos que habrá de ambos tamaños. Para ello utilizamos las variables "*g1*" y "*g2*" que mediante distintas operaciones serán las encargadas de indicar cuantas veces repetimos cada uno de los barridos.

A continuación nos centraremos en calcular el rango de escaneo que debe tener cada barrido. Esto es importante para evitar la duplicidad de datos y evitar posibles errores de localización. Para ello vemos en primer lugar la amplitud de cada escaneo en función del número de barridos. Posteriormente calculamos para esa amplitud el valor de inicio ("*amp1*") y de final ("*amp2*") que tendrá el servo en cada uno de los barridos.

Veamos un ejemplo para entenderlo mejor. Si establecemos que queremos un escaneo inicial realizado en 3 barridos, estableceremos los valores en:

- gradoGI1 = 34
- gradoGI2 = 35
- g1 = 2
- g2= 1
- amplitud = 120
- amp1 = 30
- amp2 = 150

Estos resultados indican que la función hará un total de 3 barridos: 2 de ellos tendrán un giro de 34 contadores (o 119°) y otro de 35 (122.5°). Cada barrido tendrá una amplitud de 120° que comenzara con el servo en 30° y que finalizará en 150°.

Una vez establecidos todos los parámetros es el momento de comenzar con el movimiento del robot. Para ello nos basamos en las funciones "*MServo()*" y "*giro()*" de tal modo que realice el escaneo con la amplitud y giro por barrido definidos previamente.

Para finalizar guardamos en "*giroIn*" el ángulo en el que se encuentra la distancia mínima. Nos apoyamos en la variable de "*barrido*" para calcularlo. Ésta es una variable global que usará este valor en la siguiente ecuación para orientarse.

5.3.2.2.3. Selección de camino

El siguiente punto que debemos ver es el camino que debe elegir el robot en función de los datos recogidos en las otras funciones. Para ello utilizamos los datos guardados en las tres variables importantes: "*minDistanciaIR*", "*minAnguloSv*" y "*giroIn*".

Para poder calcular este camino tenemos que tener en cuenta dos tipos de ángulos:

- El ángulo del servo en el que se ha encontrado la distancia mínima.
- El ángulo de giro del robot.

Debemos trasladar el ángulo del servo al ángulo de giro para conseguir que el robot se acabe colocando de forma rectilínea a la pared más próxima. Para calcular este ángulo de giro acudiremos a la trigonometría *(Figura 78)*.

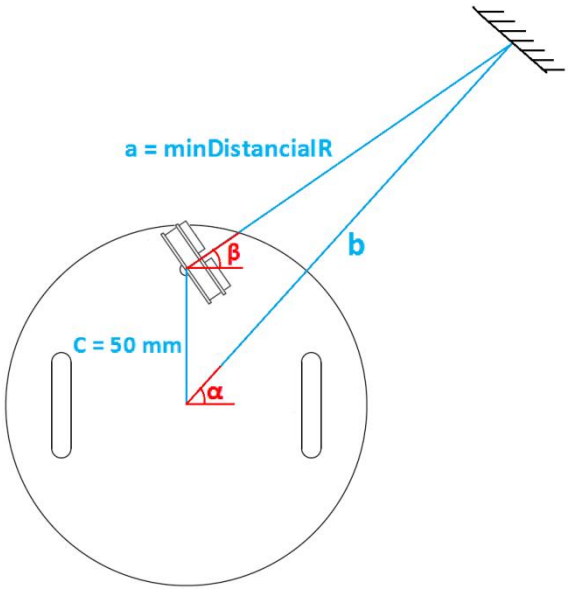


Figura 78: Triángulo servo-pared-centro robot

En la imagen anterior podemos observar el triángulo que se formaría entre el centro del robot, el ángulo del servo y la pared más cercana al robot.

Nuestro objetivo es calcular el valor del ángulo α para así poder girar el robot y colocarlo de frente a la pared más cercana. Para calcular ese ángulo necesitamos previamente conocer el valor del lado "*b*".

Para calcular ambos valores utilizaremos los teoremas del seno y del coseno.

- Teorema del Seno $\quad \frac{a}{\sin\alpha} = \frac{b}{\sin\beta} = \frac{c}{\sin\gamma}$
- Teorema del Coseno $\quad \begin{cases} a^2 = b^2 + c^2 - 2bc * \cos\alpha \\ b^2 = a^2 + c^2 - 2ac * \cos\beta \\ c^2 = a^2 + b^2 - 2ab * \cos\gamma \end{cases}$

En total son tres ecuaciones para cada uno de los teoremas. Nosotros escogeremos en nuestro caso aquellas fórmulas que utilicen los ángulos α y β.

Puesto que el servo tiene un ángulo variable que comprende entre 0° y 180°, tendremos un total de tres posibilidades en función de donde mire el sensor de infrarrojos *(Figura 79)*. Veamos cada uno de ellos.

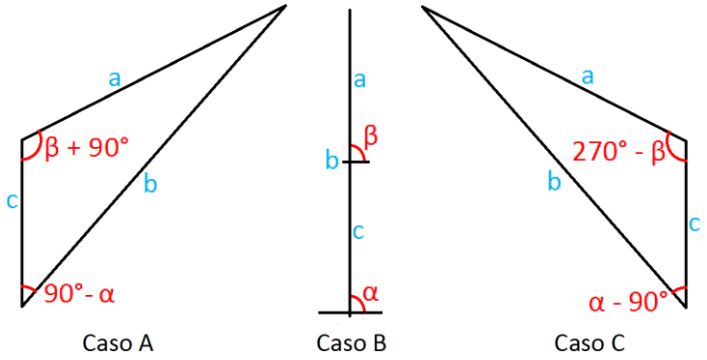


Figura 79: Casos triángulo servo-pared-centro robot

- Caso A: β < 90°

  Aquí se encuentran los ángulos en los que el servo se encuentra mirando hacia la derecha *(Primera imagen de la Figura 79)*; corresponde con todos aquellos ángulos agudos. Las ecuaciones para este primer caso son:

$$b^2 = a^2 + c^2 - 2ac * \cos(\beta + 90) \;\rightarrow\; b = \sqrt{a^2 + c^2 - 2ac * \cos(\beta + 90)}$$
$$\frac{a}{\sin(90 - \alpha)} = \frac{b}{\sin(90 + \beta)} \;\rightarrow\; \alpha = 90 - \sin^{-1}\left(\frac{a * \sin(90 + \beta)}{b}\right)$$

- Caso B: β = 90°

  Este es el caso más simple que solo ocurrirá cuando el servo se encuentre en línea con la pared y con el centro del robot *(Segunda imagen de la Figura 79)*.

Sus ecuaciones son:

$$b = a + c$$
$$\alpha = 90°$$

- Caso C: β > 90°

Este último caso corresponde con aquellos valores en los que el servo mira hacia la izquierda *(Tercera imagen de la Figura 79)* y por tanto su ángulo es obtuso. Las ecuaciones son por tanto:

$$b^2 = a^2 + c^2 - 2ac * \cos(270 - \beta) \rightarrow b = \sqrt{a^2 + c^2 - 2ac * \cos(\beta - 270)}$$
$$\frac{a}{\sin(\alpha - 90)} = \frac{b}{\sin(270 - \beta)} \rightarrow \alpha = 90 + \sin^{-1}\left(\frac{a * \sin(270 - \beta)}{b}\right)$$

Una vez tenemos calculado el ángulo α que debe girar el robot, lo único que queda es realizar el giro del robot hacia esa pared más cercana. Veamos cómo trasladamos esta idea a la función de Arduino *(Figura 80)*.

```
int direccionInicial(){
  int a=minDistanciaIR*10;      int alfa;
  int b;
  int c=50;                     int beta=minAnguloSv;

  if (beta < 90){
    b = sqrt( a*a + c*c - 2*a*c*cos((90 + beta)*DEG_TO_RAD));
    alfa = 90 - asin((a*sin((90 + beta)*DEG_TO_RAD)/b))*RAD_TO_DEG + 0.5;
  }
  if (beta > 90){
    b = sqrt( a*a + c*c - 2*a*c*cos((270 - beta)*DEG_TO_RAD));
    alfa = 90 + asin((a*sin((270 - beta)*DEG_TO_RAD)/b))*RAD_TO_DEG + 0.5;
  }
  if (beta == 90){
    b = a + c;
    alfa = 90;
  }

  int orientacion = giroIn*3.5 + alfa;

  if (0 <= orientacion && orientacion < 90){
    orientacion = 90 - orientacion;
    giro(orientacion,'R');
  }
  if (90 <= orientacion && orientacion <= 270){
    orientacion = orientacion - 90;
    giro(orientacion,'L');
  }
  if (270 < orientacion && orientacion <= 360){
    orientacion = 360 - orientacion + 90;
    giro(orientacion,'R');
  }
}
```

Figura 80: Función de selección de camino

Como se puede observar los tres casos para calcular α dependen del valor obtenido por "*minAnguloSv*" en la función anterior.

Tras esto obtenemos un nuevo ángulo referido al centro del robot y que corresponde con la suma de α y del ángulo de barrido "*giroIn*". A continuación el robot girará hacia un lado u otro (buscando el giro más corto) en función del ángulo de orientación final.

5.3.2.2.4. Movimiento inicial. Acercamiento de pared

Una vez tenemos al robot posicionado de forma correcta en frente de la pared a la que nos queremos colocar, lo siguiente será acercarse a ella y colocarse de forma paralela para que posteriormente pueda recorrerla a lo largo de la habitación.

Para simplificar los cálculos hemos decidido que el robot se coloque siempre con la rueda derecha más cerca de la pared y que avance en ese sentido. Veamos cómo es la función para poder explicarla mejor *(Figura 81)*.

```
int acercamientoPared(){
  int Izdo;    int Dcho;    int IR0;
  for (int i=0; i=100; i++){
    Izdo = readUltrasonicSensor('L');
    Dcho = readUltrasonicSensor('R');
    IR0 = medidaIRSensor();
    delay(250);
    if (Izdo != 16){
      if (Dcho != 16){
        break;
      }
    }
  }
  avanceDistancia((IR0-10),64,51);
  int girar;      int servol;       int IR1;
                                    int IR2;
  if (Izdo < Dcho){
    girar = 180;
    servol = 135;
  }
  if (Dcho <= Izdo){
    girar = 0;
    servol = 45;
  }
  int a;
  do {
    do {
      delay(250);
      IR1 = medidaIRSensor();
    }
    while (IR1 == 0);
    servoIR.write(servol);
    do {
      delay(250);
      IR2 = medidaIRSensor();
    }
    while (IR2 == 0);
    a = sqrt( IR1*IR1 + IR2*IR2 - 2*IR1*IR2*cos(45*DEG_TO_RAD));
  }
  while (IR2/a >= 1.4);
  int beta = abs(girar - asin((IR2*sin(45*DEG_TO_RAD)/a))*RAD_TO_DEG) + 0.5;
  giro(beta,'L');
}
```

Figura 81: Función de acercamiento de pared

En primer lugar utilizamos los sensores para saber con mayor precisión donde está la pared. Utilizamos los sensores de ultrasonidos para saber qué lado del robot está más próximo a la pared. El sensor de infrarrojos lo usamos con el servo en posición de 90° mirando al frente para saber a qué distancia se encuentra la pared.

Tras esto el robot avanza hasta colocarse a una cierta distancia con suficiente margen para que no se choque con la pared.

Posteriormente debemos hacer que el robot gire para que se coloque de forma paralela a la pared. Es por eso que comparamos las medidas de los sensores de ultrasonidos entre sí. En función de qué valor sea menor estableceremos unas variables que tendrán un valor u otro según el caso.

La idea es hacer que el robot en la posición actual tome una nueva medida de la pared para saber su distancia actual. A continuación el servo gira un ángulo de 45° a izquierda o derecha (en función del valor más pequeño de los sensores de ultrasonidos) y toma una nueva medida. Estos dos valores serán los que usemos para poder calcular el ángulo de giro del robot.

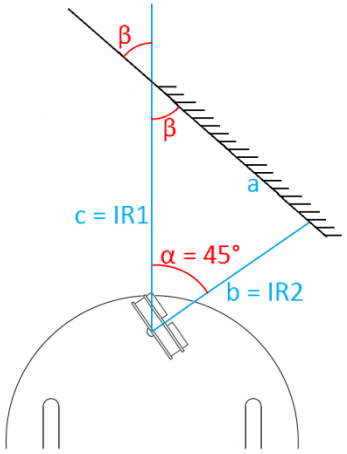


Figura 82: Ángulo de giro de pared

En la imagen *(Figura 82)* podemos observar las dos medias del sensor de infrarrojos "*b*" y "*c*"; junto con *α* volvemos a utilizar los teoremas del seno y del coseno para calcular en este caso el ángulo β que será el ángulo necesario que debe girar el robot para colocarse paralelo a la pared. Calculamos en primer lugar el lado "*a*" con el teorema del coseno y posteriormente el ángulo con el teorema del seno.

Para obtener un resultado coherente de β imponemos como condición que la división entre los lados "*b*" y "*a*" no supere el valor de 1.4 (es un modo de prever las posibles medidas erróneas tomadas por el sensor de infrarrojos). Si no se hiciese así el resultado que obtendríamos al final para el ángulo sería 0 y el robot no giraría.

Una vez calculado el ángulo solo queda mandar al robot girar el valor necesario para que se coloque de forma paralela a la pared.

5.3.2.2.5. Valor IR de referencia

Una vez tenemos posicionado al robot de forma paralela a la pared, necesitamos diseñar una manera de que el robot se mantenga siempre a la misma distancia de la pared para que no se desvíe de la trayectoria.

Es por eso que necesitamos tener una referencia para que el robot la intente seguir según avanza. Veamos la función *(Figura 83)*.

```
int refer;
int referencia(){
  servoIR.write(0);
  int i=0;
  do {
    refer=readIRSensor();
    delay(100);
    i++;
  }
  while (i<10);
  Serial.print("refer: "); Serial.println(refer);
}
```

Figura 83: Función de referencia

Para lograr este objetivo lo que haremos será colocar el sensor de infrarrojos mirando continuamente a la pared a la que va paralela; así el servo tomará un valor de 0°.

Posteriormente tomamos la medida a la que se encuentra en ese momento el robot y lo almacenamos en una variable global. A partir de ahora el robot cada vez que se mueva lo hará comparando la medida actual a la que se encuentra respecto a la pared con el valor de referencia.

5.3.2.2.6. Lectura de sensores

La siguiente función que vamos a ver es la de la lectura de los sensores *(Figura 84)*. Tras haber definido una medida de referencia el siguiente paso consiste en medir las distancias proporcionadas por los distintos sensores.

```
int IR; int USD; int USI;
int tiempoEscaneo = -1000; unsigned long tiempoRef;
int leerSensores(){
    USD = readUltrasonicSensor('R');
    IR = readIRSensor();    //medidaIRSensor();
    USI = readUltrasonicSensor('L');
    Serial.print("IR: "); Serial.println(IR);
    Serial.print("USD: "); Serial.println(USD);
    Serial.print("USI: "); Serial.println(USI);
    Serial.println();
}
```

Figura 84: Función de lectura de sensores

Declaramos una serie de variables generales que aparecerán en diversas funciones y que serán necesarias para los diversos cálculos con los que tendremos que hacer frente.

- "*IR*": esta variable corresponde al valor que recibimos del sensor de infrarrojos. Nos indicará la distancia que ha captado el sensor cada vez que llamemos a la función.
- "*USD*" y "*USI*": al igual que la variable anterior, éstas nos indican la distancia captada por los otros dos sensores. Corresponden con los sensores de ultrasonidos derecho e izquierdo, respectivamente.
- "*tiempoEscaneo*" y "*tiempoRef*": en este caso nos encontramos ante dos variables de tiempo. Nos permitirán calcular posteriormente el tiempo que pasa entre un ciclo y otro, y con ello saber lo que avanza el robot entre las distintas lecturas. La variable "*tiempoRef*" será la encargada de llevar la cuenta del tiempo en cada momento. La otra recibe inicialmente un valor negativo para poder compararlas entre sí; ya que el tiempo comienza a contar a partir de 0. Veremos su uso más adelante.

Esta función no es más que una llamada a los tres sensores del robot para que tomen una medida y que guarden su valor para poder compararlo más adelante con las otras funciones.

Opcionalmente podemos decidir si queremos mostrar el valor que toman los sensores cada vez que los llamemos mediante el uso de los comandos de "*print*"; de mostrarse debemos tener en cuenta que una gran cantidad de estos comandos utilizan buena parte de la memoria de nuestra placa de Arduino.

5.3.2.2.7. Avance y corrección de velocidad. Caso 1

El avance del robot se puede resumir en tres movimientos: avance, giro a la derecha y giro a la izquierda.

Para posteriormente poder realizar el mapa de la zona por la cual se mueve el robot, será necesario declarar una serie de variables *(Figura 85)* que serán comunes para los tres casos de movimiento.

```
int desvio=0;        int alfa=0;
int aTiempo;         int bTiempo;
```

Figura 85: Variables de posición del robot

Veamos de qué se tratan:

- "*aTiempo"* y *"bTiempo*": estas variables están relacionadas con el avance de las ruedas. La diferencia que se obtenga de estos valores nos permitirá calcular posteriormente cuanto se ha desplazado el robot en los tramos en los que avanza recto.

- *"desvío"*: este valor es un contador que indica las veces que el robot se ha desviado de la trayectoria recta y ha habido que corregirlo.
- *"alfa"*: esta variable adquiere un valor diferente en cada uno de los tres casos. En función del ángulo que muestre nos informará de hacia dónde gira el robot.

Debido a que el sensor de infrarrojos nos da una mayor fiabilidad que los dos sensores de ultrasonidos (ya que podemos orientar hacia donde queremos tomar la medida gracias al servo), usaremos dicho sensor mirando de forma perpendicular a la pared de tal modo que sea la referencia mediante la cual se mueva el robot.

Así nos garantizamos una mejor medida de la distancia a la que se encuentra el robot. El robot se desplazará en línea recta siguiendo la dirección de la pared.

Ya hemos comentado en alguna ocasión que los motores de las ruedas alcanzan distinta velocidad cuando se les da el mismo valor de potencia. Esto hace que el robot se desvíe continuamente hacia un lado (en nuestro caso se desvía hacia la derecha) porque el motor izquierdo gira algo más rápido que el motor derecho.

Es por esta razón por la cual el sensor de infrarrojos mira continuamente la pared *(Figura 86)*. De esta forma tenemos un mayor control sobre la trayectoria del robot y podemos saber cuándo se desvía y la forma de contrarrestarlo.

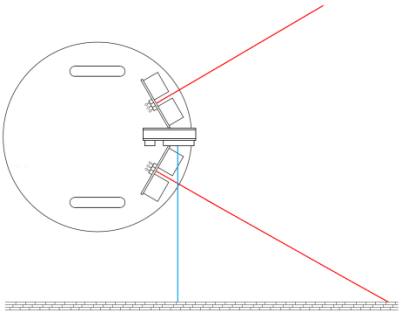


Figura 86: Caso 1. Avance

En un principio se pensó en diseñar una función con un regulador PID (hablaremos de esta función más adelante), pero debido a la baja resolución de nuestros encoders se descartó la idea.

De este modo la forma que tenemos de corregir la trayectoria del robot consiste en realizar continuas desviaciones hacia el lado contrario. Así cada vez que el robot se acerque demasiado a la pared, giraremos un poco su ángulo hacia fuera de tal modo que se aleje de la misma. Puesto que según avance el robot éste seguirá desviándose hacia la pared, las correcciones hacia fuera serán continuas. Precisamente este punto ese lo que tratamos con esta función.

En esta ocasión nos centramos en el primer caso: el avance. Puesto que a medida que avanza el robot se desvía también vemos en esta función la manera de corregirlo. Veamos en que consiste dicha función *(Figura 87)*.

```
int casol(){
  if (alfa != 0){
    aTiempo = tiempoRef;
  }
  if (IR < refer - 5){
      giro(14,'L');
      avanceDistancia(5,120,111);
      desvio++;
  }
  alfa = 0;
}
```

Figura 87: Función de avance

El robot se desplaza a velocidad constante. En nuestro caso hemos decidido que se mueva a una velocidad correspondiente al 50% de la potencia total de los motores. Por tanto el robot avanza de forma continua y solo entrará en esta función cuando se cumplan alguna de las condiciones impuestas.

La primera condición con la que nos encontramos es que alfa sea distinto de cero. Si se cumple quiere decir que el robot lo último que ha hecho ha sido realizar un giro (bien a izquierda o derecha). El valor de alfa como hemos dicho antes varía en función del caso en el que nos encontremos. Si en el ciclo anterior el robot ha girado lo que conseguimos con esta condición es inicializar la variable de tiempo "*aTiempo*" al valor actual del contador (que viene dada por "*tiempoRef*"). De este modo cuando el robot vuelva a girar en ciclos posteriores, guardaremos en ese momento el valor del contador en la variable "*bTiempo*". La diferencia entre ambas variables nos indicará posteriormente el tiempo que ha permanecido el robot avanzando en línea recta.

La segunda condición tiene que ver con el valor de referencia que tengamos en cuenta a la hora de comparar la distancia que existe entre el robot y la pared. Este valor de referencia ("*refer*") viene dado por la función "*referencia()*" que vimos anteriormente. Del mismo modo podemos elegir también declarar un valor fijo a esta referencia (en nuestro caso por lo general "*refer*" tendrá un valor de 15 cm con respecto a la pared).

De cualquier forma, está condición compara ese valor de referencia con el valor que en ese instante recibe el robot del sensor de infrarrojos, que como hemos dicho corresponde con "*IR*". En el momento en el que detecte que el valor del sensor de infrarrojos es menor que la referencia más un pequeño margen (5 cm en nuestro caso), significa que el robot se ha desviado de su trayectoria y se acerca a la pared. Por ello al entrar en esta condición el robot gira un pequeño ángulo y avanza una pequeña distancia en esa dirección para garantizar que el robot se aleja lo suficiente del umbral del valor de referencia. Esto hace que también el contador de desvío aumente cada vez que se produce este caso; de este modo podremos usarlo posteriormente para establecer las distancias recorridas.

Al final de la función establecemos que alfa sea cero. Con ello indicamos que el robot no ha girado y que se avanza en línea recta. Independientemente de sí el robot ha entrado previamente en alguna de las dos condiciones indicadas anteriores, al final del "*caso1()*" siempre indica que el ángulo es cero y que de avanzamos en línea recta.

Puesto que nos centramos en mantener paralelo al robot mediante el sensor de infrarrojos, los sensores de ultrasonidos aquí no son necesarios. La única condición importante tiene que ver con el sensor de infrarrojos.

5.3.2.2.8. Giro a la derecha. Caso 2

El siguiente caso a tratar consiste en el giro del robot hacia la derecha. En esta ocasión si tendremos en cuenta aparte del sensor de infrarrojos el sensor de ultrasonidos derecho que usaremos como apoyo para esta ocasión.

Necesitamos saber en esta ocasión cuando el robot puede girar al llegar a una esquina.

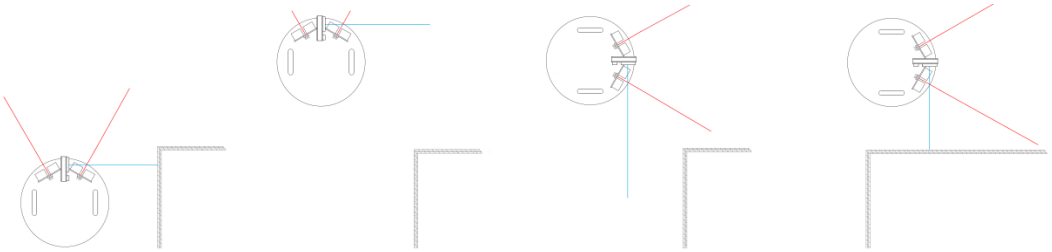


Figura 88: Caso 2. Giro a la derecha

Si observamos la imagen *(Figura 88)* podemos ver que cuando el robot llega a este tipo de esquinas el sensor de infrarrojos pasa de recibir un valor de pared próximo al de referencia a recibir otro con una distancia mucho mayor.

Este hecho es el que nos indica que nos encontramos ante el segundo caso. Por ello cuando el robot detecta esta situación debe avanzar lo suficiente para evitar la esquina y después girar 90° hacia la derecha para así volver a estar en paralelo con la pared.

```
int caso2(){
  if (IR > refer + 10){
    avanceDistancia(18,120,111);
    bTiempo = tiempoRef;
    IR = medidaIRSensor();
    if (IR > refer + 10){
      giro(87.5,'R');
      delay(500);
      int IRcaso2;
      do{
        setMotorVal(60,51);
        int USDcaso2 = readUltrasonicSensor('R');
        IRcaso2 = readIRSensor();
        if (USDcaso2 < 10){
          giro(21,'L');
        }
      }
      while (IRcaso2 > refer + 10);
    }
    alfa = -90;
  }
}
```

Figura 89: Función de giro a la derecha

Al igual que en el caso anterior, para que se active esta función *(Figura 89)* es necesario que se cumpla la condición de igualdad. En esta ocasión tomamos la medida con el sensor de infrarrojos y comprobamos si este valor supera al de referencia en una distancia considerable; de ser así entonces el robot supone que la pared se ha acabado en esa dirección.

A continuación indicamos al robot que siga avanzando una distancia lo suficientemente grande como para que todo el robot haya cruzado la esquina de la pared y tenga margen de maniobra.

Asimismo guardamos en ese momento el valor del contador de tiempo que viene dado por "*tiempoRef*". Este es el segundo valor que necesitamos para poder calcular la distancia avanzada hasta el momento del giro junto con "*aTiempo*".

Tras esto volvemos a comprobar con otra condición que efectivamente el robot sigue devolviendo un valor bastante mayor al de referencia. Si esto es así, giramos 90° hacia la derecha de tal modo que se coloque paralelo a la nueva pared. En este punto el sensor de infrarrojos sigue detectando una distancia mucho mayor que la de referencia. Esto le indica al robot que aún no se ha encontrado con la nueva pared y que debe seguir avanzando en esa dirección hasta que se aproxime.

Por tanto hacemos que el robot continúe avanzando hasta que volvamos a tener la condición de que el sensor de infrarrojos vuelve a ser menor que el de referencia. Por ello vamos tomando distintas medidas según se mueve para así compararlo y ver cuando se cumple.

Si nos aproximamos demasiado a la pared antes de que el sensor de infrarrojos lo detecte por estar en un punto ciego o bien por el desvío de las ruedas, tenemos para ello el sensor de ultrasonidos derecho que nos indicará cuando su valor es muy pequeño y que el robot se encuentra avanzando en una posición errónea y que la pared está próxima. Por ello que el robot debe corregir el ángulo para evitar chocarse.

En el caso de que la esquina no tuviese un ángulo recto; sino que fuese un ángulo mayor o menor, el robot actuaría de la misma forma solo que en esta ocasión llegaría antes (si el ángulo es obtuso) o después (si el ángulo es agudo) a la pared debido al desvío de los motores. En el momento en el que volviese a sobrepasar el valor de referencia entraríamos en el primer caso y se corregiría.

Finalmente establecemos el valor de alfa a -90 ° para indicar que el robot ha girado hacia la derecha.

5.3.2.2.9. Giro a la izquierda. Caso 3

Este es el tercer caso que nos podemos encontrar cuando el robot avanza por una zona. En esta ocasión nos centraremos en comparar los sensores de ultrasonidos.

Aquí debemos girar hacia la izquierda puesto que el robot de seguir avanzando se encontraría de frente contra la pared *(Figura 90)*.

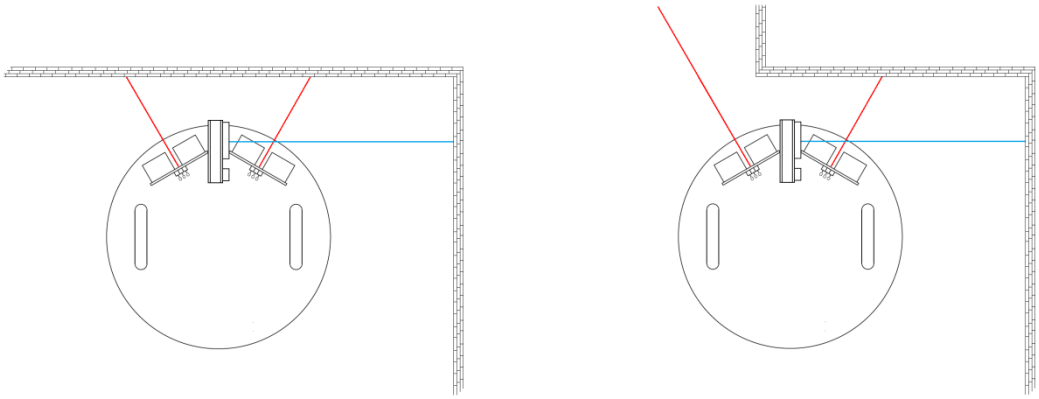


Figura 90: Caso 3. Giro a la izquierda

Podemos encontrarnos con dos mecánicas distintas, pero el tratamiento es el mismo. Solo varían las condiciones.

- El robot puede encontrarse con una pared de frente que le obliga a girar a la izquierda. Esta pared es lo suficientemente grande para que ambos sensores de ultrasonidos lo detecten.
- El robot puede encontrarse con un pequeño saliente de tal forma que el sensor derecho sí detecta la pared pero no ocurre lo mismo con el izquierdo.

Veamos en esta ocasión cómo quedaría la función *(Figura 91)*.

```
int caso3(){
  if (USD < refer){
    if (USI < refer || USI > refer + 10){
      bTiempo = tiempoRef;
      giro(87.5,'L');
      delay(500);
      alfa = 90;
    }
  }
}
```

Figura 91: Función de giro a la izquierda

En esta ocasión establecemos como condición principal que el sensor de ultrasonidos derecho detecte un valor menor que el de referencia. Esto ya le indica al robot que se va a encontrar con una nueva pared de frente. Cuando el robot se encuentre con este caso, entrará en la función.

Cuando el robot está lo suficientemente cerca como para que ambos sensores de ultrasonidos detecten valores similares, entonces giramos 90° hacia la izquierda.

Lo mismo ocurre si cuando entra en la función el robot detecta que el sensor derecho está muy próximo a la pared y el izquierdo sin embargo obtiene un valor mucho mayor, puesto que no hay muro por tratarse de un pequeño saliente.

Del mismo modo que en la función anterior guardaremos el valor final del contador de tiempo para calcular más adelante la distancia y además estableceremos el valor de alfa a 90° para indicar en esta ocasión que el giro se ha producido hacia la izquierda.

5.3.2.2.10. Guardado de datos

Una vez hemos visto los distintos casos en los cuales se mueve el robot, es necesario reflejar dichos movimientos para tener un seguimiento de la zona por la cual se mueve.

Esta parte está destinada a la creación de distintos mapas que se van generando con los sensores a medida que el robot avanza.

Se han planteado dos funciones distintas en base al modo de utilizarlas:

- Una función crea los mapas a medida que el robot avanza paralelo a las paredes, guardando los valores obtenidos por los sensores.
- La otra función crea los mapas con el robot colocado en el mismo punto. En esta ocasión el robot no se mueve, va girando sobre sí mismo y creando los mapas con los datos de alrededor.

Para poder entender cómo funcionan ambas funciones es importante tener claro cómo están planteadas.

Para representar un mapa 2D del entorno del robot en primer lugar debemos plantearnos un sistema de coordenadas global en el cual nuestro robot se irá desplazando. Este sistema de coordenadas es el principal y será al cual referiremos el resto de datos.

Nuestro robot a su vez tendrá su propio sistema de coordenadas local en el que tendremos unos valores que serán distintos cuando sean llevados al sistema de coordenadas global *(Figura 92)*.

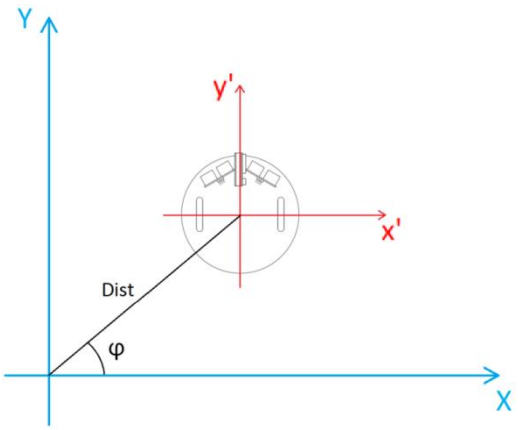


Figura 92: Sistemas de coordenadas global y local

Para poder representar en un mapa una posición son necesarios dos datos: distancia y ángulo.

Con ellos podemos obtener los valores X e Y necesarios para marcar su posición en el mapa.

$$X_{robot} = Dist * \cos\varphi$$
$$Y_{robot} = Dist * \sin\varphi$$

Para obtener el valor de la distancia utilizaremos como ya hemos indicado anteriormente los intervalos de tiempo que tarda el robot en recorrer cada uno de los tramos rectos. Estos tramos comienzan desde el último giro del robot hasta que se produce el siguiente.

Para saber el ángulo bastará con saber hacia dónde gira el robot utilizando los contadores de los encoders.

En el caso en el que el robot escanee la habitación sin moverse por la estancia, la distancia será cero; pero en el caso de que vayamos creando el mapa según se mueve tendremos en cuenta desplazamientos en línea recta y giros de 90°.

Veamos en primer lugar como hallar la posición. El robot parte inicialmente del punto [0,0] respecto del eje de coordenadas global. A medida que los motores se mueven, iremos llevando la cuenta del tiempo que transcurre.

Para saber la distancia recorrida aparte del tiempo que tarda en recorrer el robot un cierto espacio necesitamos otro dato. Este dato corresponde con la velocidad. Con ambos podemos calcular fácilmente el valor de las distancias.

Por lo tanto inicialmente el robot se movería en la dirección [0,Y]. Esto significa que el robot comenzaría con un ángulo φ de 90°.

Puesto que suponemos que el robot se mueve en línea recta, descartamos todas aquellas posiciones del robot que se encuentren en la misma dirección y sentido que el último punto obtenido. Solo tendremos en cuenta los puntos en los que se ha cambiado de dirección.

En el momento en el que el robot gire hacia izquierda o derecha tendremos los datos necesarios para calcular el siguiente punto.

Tras ello el robot volverá a avanzar y en el momento en el que vuelva a girar se calculará un nuevo punto; sin embargo en esta ocasión el ángulo de giro serán los 90° iniciales más el ángulo del primer giro (+90° si gira a la derecha o -90° si es hacia la izquierda). Este proceso lo podemos ver en la siguiente imagen *(Figura 93).*

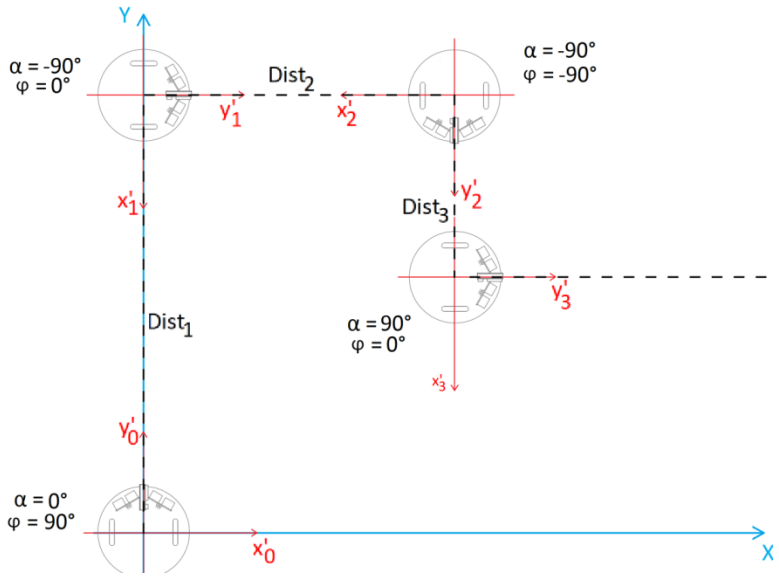


Figura 93: Representación de posición del robot

Podemos decir entonces que el robot realiza giros locales de ±90° (representados mediante α) mientras que globalmente el giro de cada uno de los puntos es la acumulación de todos los giros locales que ha realizado hasta ese momento.

$$\varphi = \varphi + \alpha$$

De este modo podemos obtener poco a poco el mapa de la zona en la cual nos movemos.

Una vez tenemos la posición del robot en los puntos clave, lo siguiente es crear los mapas a partir de los datos obtenidos por los sensores. Cada sensor mostrará una distancia en función del obstáculo que tenga delante. Estas distancias se encuentran referidas en los ejes locales, puesto que los sensores están situados en el robot.

Al igual que antes para determinar la posición, deberemos trasladar las distancias en ejes locales de los sensores a su situación en ejes globales. Para ello debemos conocer su posición relativa respecto al robot. Cada sensor se encuentra situado en un ángulo determinado β *(Figura 94)*.

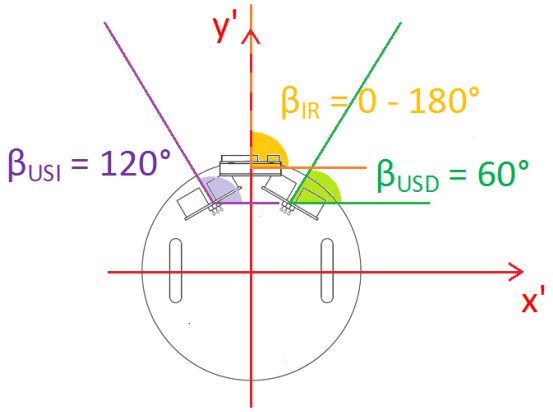


Figura 94: Posiciones locales de los sensores

El sensor de infrarrojos al estar situado encima del servo tiene un ángulo variable que dependerá de su posición en cada caso. Los sensores de ultrasonidos tienen una posición fija.

Por lo tanto para hallar la posición de los objetos detectados por los sensores deberemos añadir esta distancia a la posición del robot en ejes globales. Para ello primero debemos calcular las coordenadas en ejes locales de la distancia de los sensores:

$$x_{sensor} = dist_{sensor} * \cos\beta$$
$$y_{sensor} = dist_{sensor} * \sin\beta$$

Es importante indicar que como estas distancias son locales, también están sujetas a los giros locales α que pueda realizar el robot. De este modo tenemos también que:

$$\beta = \beta + \alpha$$

Finalmente tenemos que la posición en ejes globales marcada por los sensores es:

$$X = X_{robot} + x_{sensor}$$
$$Y = Y_{robot} + y_{sensor}$$

Otro detalle que debemos tener en cuenta en este caso, es la forma en la que calculamos el valor de la distancia avanzada por el robot.

Como hemos dicho antes para calcularlo necesitamos dos datos distintos:

- El tiempo que tarda en recorrer una distancia.
- La velocidad de movimiento del robot.

Para conocer el valor de la velocidad necesitamos medirla de alguna forma. Es por ello que usamos las funciones de "*velMotorI()*" y "*velMotorD*" y conocer así la velocidad de los motores para cada valor de potencia que le demos.

De este modo tenemos que la distancia que obtenemos es:

$$Dist = v * t$$

Otra forma de obtener el valor de la distancia es mediante los contadores. Llevando la cuenta del número de veces que se activan los contadores a medida que avanzan las ruedas podemos saber directamente la distancia recorrida para cada uno de los tramos.

Aunque es un buen método para calcular distancias, finalmente decidimos descartar esta opción por la carga que le ocasiona a nuestro programa de Arduino. Debido a las limitaciones de nuestro modelo de placa, los valores de distancia que obteníamos eran

mucho menores de lo que en realidad eran. Esto se producía cuando el robot debía llevar la cuenta a la vez de los sensores de infrarrojos y de ultrasonidos además de las distancias de los contadores.

El robot no es capaz de realizar a la vez todas las operaciones que se le piden y esto se traduce en perdida de resolución en los contadores. Para futuras ampliaciones de la memoria del robot tendremos en cuenta esta opción para implementarla en lugar de recurrir a un contador de tiempo.

Una vez visto el primer caso, pasemos a explicar la teoría del segundo tipo de mapeado.

El segundo tipo de mapeo es más sencillo; en el caso en el que se realice girando desde el eje [0,0] los valores X e Y del robot serán 0 y el giro relativo de α dependerá del ángulo que le queramos dar en cada incremento.

Por tanto las distancias de movimiento del robot serán 0 y solo tendremos en cuenta las distancias de los sensores. Para los ángulos tendremos que sumar dos tipos de ángulos: los ángulos β de cada sensor con el ángulo correspondiente al giro que realice el robot α.

En este caso el robot realiza una seria de pequeños giros α de igual amplitud que se van acumulando hasta completar una vuelta de 360°. Será necesario sumar a este ángulo acumulado en cada medida el valor del ángulo β según el sensor que tengamos.

Tras haber visto la teoría de cómo representamos los valores en un mapa, es el momento de llevarlo a cabo en las funciones de Arduino. Ambas funciones necesitarán de unas variables que serán comunes casi en su totalidad *(Figura 95)*.

```
int phi=90;                int betaIR=0;              int betaUSD=60;             int betaUSI=120;
int xP=0;                  int yP=0;
int iM=0;                  int iS=0;
int matrizPosicion[2][20]; int matrizSensorIR[2][20]; int matrizSensorUSD[2][20]; int matrizSensorUSI[2][20];
int incrAvance = -42;      int Dist;                  int IntroducirDato = 0;
```

Figura 95: Variables de mapeo

Veamos en qué consisten:

- *"phi"*: este ángulo corresponde con la posición en la que se encuentra el robot respecto a un eje de coordenadas. Su valor indica la dirección que sigue en su avance.
- *"betaIR"*, "*betaUSD*" y "*betaUSI*": corresponden a los ángulos de los distintos sensores con respecto al robot.
- "*xP*" e "*yP*": son las coordenadas X e Y del robot en cada momento.
- "iM" e "iS": son contadores destinados a llevar un orden de los valores que se van almacenando en las matrices.
- "*matrizPosicion[][]*", "*matrizSensorIR[][]*", "*matrizSensorUSD[][]*" y "*matrizSensorUSI[][]*": son las matrices en las cuales vamos a ir guardando los

datos correspondientes a la posición del robot y de los sensores. Dichas matrices constan de dos filas; una para las posiciones de X y otra para las posiciones de Y.

- "*incrAvance*": este valor indica lo que avanza el robot en línea recta en cada incremento.
- "*Dist*": es la distancia recorrida por el robot en cada uno de los tramos rectos.
- "*IntroducirDato*": esta variable indicará el momento en el que puede introducir un nuevo dato a las matrices.

Tras explicar cuáles van a ser nuestras variables principales, es el momento de centrarnos en las funciones en sí. Veamos en primer lugar la función que realiza el mapeo de la zona según se va moviendo el robot *(Figura 96)*.

```
int matrizDatos(){
  if (alfa != 0){
    Dist = 11.22*abs(bTiempo - aTiempo)/1000 - desvio*1.1;
    int xa = Dist * cos(phi*DEG_TO_RAD) + 0.5;
    int ya = Dist * sin(phi*DEG_TO_RAD) + 0.5;
    xP = xP + xa;
    yP = yP + ya;
    iM++;
    matrizPosicion[0][iM] = xP;
    matrizPosicion[1][iM] = yP;
    phi = phi + alfa;
    desvio=0;
    incrAvance = 0;
  }

  matrizPosicion[0][0] = 0;
  matrizPosicion[1][0] = 0;


  if (alfa != 0 || IntroducirDato == 1){

    int xsIR = IR * cos (betaIR*DEG_TO_RAD) + incrAvance * cos(phi*DEG_TO_RAD) * 0.2748;
    int ysIR = IR * sin (betaIR*DEG_TO_RAD) + incrAvance * sin(phi*DEG_TO_RAD) * 0.2748;

    int xsUSD = USD * cos (betaUSD*DEG_TO_RAD) + incrAvance * cos(phi*DEG_TO_RAD) * 0.2748;
    int ysUSD = USD * sin (betaUSD*DEG_TO_RAD) + incrAvance * sin(phi*DEG_TO_RAD) * 0.2748;

    int xsUSI = USI * cos (betaUSI*DEG_TO_RAD) + incrAvance * cos(phi*DEG_TO_RAD) * 0.2748;
    int ysUSI = USI * sin (betaUSI*DEG_TO_RAD) + incrAvance * sin(phi*DEG_TO_RAD) * 0.2748;

    matrizSensorIR[0][iS] = matrizPosicion[0][iM] + xsIR;
    matrizSensorIR[1][iS] = matrizPosicion[1][iM] + ysIR;

    matrizSensorUSD[0][iS] = matrizPosicion[0][iM] + xsUSD;
    matrizSensorUSD[1][iS] = matrizPosicion[1][iM] + ysUSD;

    matrizSensorUSI[0][iS] = matrizPosicion[0][iM] + xsUSI;
    matrizSensorUSI[1][iS] = matrizPosicion[1][iM] + ysUSI;

    iS++;
    betaIR = betaIR + alfa;
    betaUSD = betaUSD + alfa;
    betaUSI = betaUSI + alfa;

  }
}
```

Figura 96: Función de guardado de datos 1

Es importante aclarar que la función depende directamente de las funciones de movimiento de los casos 1,2 y 3. Ya que son ellas las que nos indican el valor del ángulo α; fundamental para que esta función realice una u otra acción.

Como se puede comprobar lo primero que nos encontramos es un bucle "*if"* cuya condición de entrada es que α sea distinto de 0. Cuando esto ocurre le estamos indicando a la función que el robot ha girado (o bien a izquierda o bien a la derecha); por lo que debemos guardar esa posición.

Para guardar la posición primero calcula la distancia. Esto lo hace con unas simples operaciones. En primer lugar tenemos que saber el tiempo que tarda en recorrer los trayectos en línea recta:

- "*aTiempo*" se guarda la primera vez que el robot avanza en línea recta tras haber realizado un giro.
- "*bTiempo*" guarda el valor del tiempo cada vez que se produce un giro.

En segundo lugar debemos conocer la velocidad a la que se desplaza el robot. En nuestro caso el robot se desplaza con una potencia del 50% del máximo que dan de sí los motores. Mediante las funciones de velocidad explicadas anteriormente podemos saber qué velocidad le corresponde.

Tras realizar distintas pruebas comprobamos que la velocidad que alcanza el robot para esa potencia es de 51 rpm. Tendremos que pasar ese valor a cm/s. Para ello en primer lugar tendremos que obtener la velocidad en rad/s y posteriormente transformar la velocidad angular en lineal para así saber que le corresponde.

$$\omega = \frac{2\pi}{60} * [rpm] \rightarrow \frac{2\pi}{60} * 51 \rightarrow \omega = 5.34\, rad/s$$
$$v = \omega * R \rightarrow 5.34\, [rad/s] * 2.1[cm] \rightarrow v = 11.22\, cm/s$$

Como sabemos que el radio de las ruedas es de 2.1 cm, solo tenemos que aplicar al final la fórmula de la relación de la velocidad y obtenemos así un valor de 11.22 cm/s de velocidad lineal. En el caso de utilizar otra velocidad de desplazamiento el proceso sería el mismo.

Conocidos por tanto los valores de velocidad y tiempo podemos calcular la distancia recorrida.

$$s = v * t$$

También será necesario tener en cuenta la distancia de las posibles desviaciones en el caso de que se produzcan, y para ello le restaremos la distancia correspondiente a cada uno de los desvíos. Sabiendo que el desvío es de cuatro contadores, tenemos una distancia de 1.1 cm/desvío. Multiplicamos este valor por las veces que se ha salido del camino marcado y tendremos la distancia total desviada.

Tras conocer la distancia pasamos a calcular la posición relativa del robot con respecto al último punto. Para ello utilizamos el valor de la distancia y del ángulo φ que tenía hasta ese momento. Con ellos obtenemos las coordenadas X e Y en cm.

Sumamos estos valores a la última posición conocida del robot y obtenemos de esta manera un nuevo punto para la creación del mapa que guardamos en la matriz correspondiente a la posición global del robot.

Tras esto establecemos el nuevo valor del ángulo φ añadiéndole el ángulo α que ha producido que entremos en el bucle *"if"*. Del mismo modo reiniciamos los valores de "*desvío*" e "*incrAvance*". También aumentamos en 1 el contador de las posiciones de la matriz.

Tras el bucle "*if*" declaramos la primera posición del robot y la guardamos. Establecemos como posición inicial el punto [0,0] ya que será esa la posición de partida a partir de la cual se generarán los mapas.

A continuación nos encontramos con otra condición. La función entrará en ella en el caso de que el ángulo alfa sea distinto de 0 o bien que la variable de "*IntroducirDato*" se encuentre activa con un valor de 1.

De producirse alguno de estos casos, estaremos en disposición de calcular las posiciones X e Y de los obstáculos localizados por cada uno de los sensores. Para ello usamos la misma mecánica que antes salvo que en esta ocasión elegimos como distancia los valores obtenidos por los sensores.

Cada vez que se llame a la función el robot habrá avanzado una pequeña distancia y los sensores reciben un valor diferente cada vez. Por ello a la hora de calcular la posición de los sensores le sumamos un incremento correspondiente al pequeño avance producido entre dos llamadas a la función de mapeo.

Al tener tres sensores distintos, obtendremos tres mapas diferentes. Calculamos en primer lugar la posición relativa de los obstáculos con respecto al robot y posteriormente usamos esos valores para calcular su posición global en el mapa.

Utilizamos como base la matriz de posición del robot y le sumamos a ésta las posiciones locales obtenidas por los sensores para crear así unas nuevas matrices correspondientes cada una de ellas con el mapa de un sensor determinado.

Del mismo modo y tras haber guardado la última posición en la matriz, procedemos a preparar los datos para la siguiente. Con ello establecemos los nuevos valores de los ángulos. En este caso tratamos con el ángulo β que será distinto para cada uno de los sensores; sin embargo los tres casos están sujetos al mismo influjo del ángulo α cada vez que el robot realiza un giro. El contador de las matrices de los sensores también aumenta su valor en 1.

Para poder ver estos resultados por pantalla podemos incluir unas cuantas líneas de código más para así visualizar cómo se van rellenando las matrices y que valores tienen en el eje de coordenadas *(Figura 97)*.

```
Serial.print("X: ");
for(int m=0; m<iM; m++){
  Serial.print(matrizPosicion[0][m]);
  Serial.print(" ");
}
Serial.println(); Serial.print("Y: ");
for(int n=0; n<iM; n++){
  Serial.print(matrizPosicion[1][n]);
  Serial.print(" ");
}
Serial.println();

Serial.print("Xir: ");
for(int o=0; o<iS; o++){
  Serial.print(matrizSensorIR[0][o]);
  Serial.print(" ");
}
Serial.println(); Serial.print("Yir: ");
for(int p=0; p<iS; p++){
  Serial.print(matrizSensorIR[1][p]);
  Serial.print(" ");
}
Serial.println();

Serial.print("Xusd: ");
for(int q=0; q<iS; q++){
  Serial.print(matrizSensorUSD[0][q]);
  Serial.print(" ");
}
Serial.println(); Serial.print("Yusd: ");
for(int r=0; r<iS; r++){
  Serial.print(matrizSensorUSD[1][r]);
  Serial.print(" ");
}
Serial.println();

Serial.print("Xusi: ");
for(int s=0; s<iS; s++){
  Serial.print(matrizSensorUSI[0][s]);
  Serial.print(" ");
}
Serial.println(); Serial.print("Yusi: ");
for(int t=0; t<iS; t++){
  Serial.print(matrizSensorUSI[1][t]);
  Serial.print(" ");
}
Serial.println();
```

Figura 97: Función de guardado de datos 1. Resultados

A medida que obtenemos distintos puntos, las matrices van guardando y representando su posición. Para poder hacernos una idea del mapa que mostrarían bastará con trasladar esos puntos a una gráfica que los represente. Esto lo veremos más adelante en otro apartado.

Una vez hemos visto la función que crea los mapas con el robot en movimiento, es el turno de ver la otra versión en la que el robot manteniéndose en una posición fija va

realizando pequeños giros y tomando valores del entorno en el que se encuentra. Veamos en que consiste *(Figura 98)*.

```
int mapeoInicial(int Ngrados){
  int gradoMin = Ngrados/3.5 + 0.5;
  gradoMin = gradoMin*3.5;

  betaIR = 90;
  servoIR.write(90);

  for (int i=0; i<360/Ngrados; i++){
    leerSensores();

    int xsIR = IR * cos (betaIR*DEG_TO_RAD);
    int ysIR = IR * sin (betaIR*DEG_TO_RAD);
    int xsUSD = USD * cos (betaUSD*DEG_TO_RAD);
    int ysUSD = USD * sin (betaUSD*DEG_TO_RAD);
    int xsUSI = USI * cos (betaUSI*DEG_TO_RAD);
    int ysUSI = USI * sin (betaUSI*DEG_TO_RAD);

    matrizSensorIR[0][i] = xsIR;
    matrizSensorIR[1][i] = ysIR;
    matrizSensorUSD[0][i] = xsUSD;
    matrizSensorUSD[1][i] = ysUSD;
    matrizSensorUSI[0][i] = xsUSI;
    matrizSensorUSI[1][i] = ysUSI;

    betaIR = betaIR + gradoMin;
    betaUSD = betaUSD + gradoMin;
    betaUSI = betaUSI + gradoMin;

    giro(gradoMin,'L');
  }
}
```

Figura 98: Función de guardado de datos 2

En esta función nos encontramos con una variable de entrada llamada "*Ngrados*". Dicha variable corresponde con la amplitud que queremos entre giros; es decir el mínimo valor de grados que debe haber entre una medida y otra.

Una vez introducido este dato lo primero que hace la función es adaptar este ángulo a nuestro caso. Recordemos que el ángulo más pequeño que puede girar el robot es de 3.5°. Las dos primeras líneas de código nos permiten hacer precisamente eso, saber el número de contadores que equivale el ángulo introducido y convertirlo a un valor múltiplo del giro mínimo del robot. Esta nueva variable recibe el nombre de "*gradoMin*".

Tras esto y para asegurarnos de una correcta posición del sensor de infrarrojos, establecemos que el ángulo de β para este caso sea de 90° para que así capte los obstáculos que se encuentren de frente. Del mismo modo establecemos la posición del servo para este mismo ángulo.

A continuación entramos en un bucle "*for*" que será el encargado de realizar realmente la función del mapeo. La amplitud de este bucle será el número total de medidas que tomará el robot hasta dar una vuelta completa.

En primer lugar leemos el valor de los tres sensores y seguidamente calculamos sus coordenadas X e Y en función de la distancia obtenida y del ángulo β. Seguidamente guardamos estas posiciones en su correspondiente matriz.

Una vez guardados los resultados de la posición actual, calculamos los nuevos valores del ángulo β de cada uno de los sensores. Puesto que en la siguiente iteración el robot estará desplazado un incremento igual a "*gradoMin*".

Finalmente el robot gira el ángulo equivalente de "*gradoMin*" para tomar una nueva medida y repetir el proceso. Cuando el robot haya dado la vuelta completa y tenga todos los puntos guardados, el bucle "*for*" habrá terminado y con ello la función.

En esta ocasión no tenemos matriz de posición del robot; ya que aquí el robot no se desplaza, solo gira sobre sí mismo hasta realizar una vuelta completa. Por lo que las únicas matrices que utilizamos son las de los tres sensores.

Al igual que en la otra función de mapeo, podemos mostrar los resultados según se van obteniendo mediante las líneas de código de "*printf*" mostradas en la figura 97.

5.3.2.3. Funciones descartadas

En este apartado vamos a ver algunas funciones que se pensaron implementar en un principio en el programa principal pero que finalmente no han aparecido en él.

5.3.2.3.1. Función PID

Debido al problema de los motores mediante el cual a igual valor de potencia uno gira a más velocidad que el otro, se pensó que una posible manera de corregirlo sería mediante la implementación de un controlador PID *(Figura 99)*.

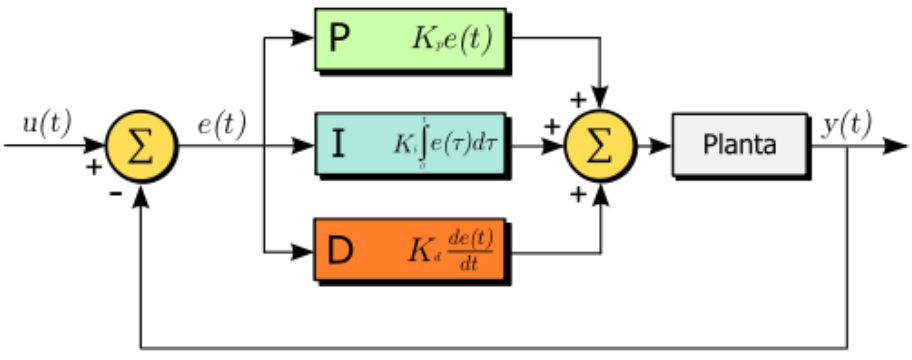


Figura 99: Diagrama controlador PID

Un controlador PID es una función dependiente del tiempo en la cual mediante sucesivas iteraciones y estableciendo los parámetros adecuados podemos alcanzar un valor concreto.

El objetivo de un controlador PID es el de minimizar el error. El algoritmo se adapta para intentar obtener un valor fijo de salida en función de distintos valores de entrada que varían con el tiempo. Una función PID consta de tres elementos principales:

- Acción proporcional: Es el producto entre una constante K_p y la señal de error.
- Acción integral: Es el producto entre una constante K_I y el error integral de tipo acumulativo.
- Acción derivativa: Es el producto entre una constante K_D y el error de tipo derivativo.

Con un PID calculamos el error existente entre el valor que queremos obtener y el valor medido. Con este error se aplica una corrección que posteriormente se utiliza en la siguiente iteración para minimizar los errores posteriores. Este proceso se realiza continuamente puesto que el sistema se retroalimenta de los resultados que va obteniendo y los utiliza para optimizar el resultado y hacer que el error sea el menor posible.

Teniendo esta idea presente se diseñó dicha función para que pudiese calcular el error cometido en los errores y poder así igualar la velocidad de ambas ruedas. Veamos en qué consistía *(Figura 100)*.

```
double errorSum=0;

int PID(int motorVal, double velObjetivo, double velMotor, float Kp, float Kd, float Ki){
  double pidSum = 0;
  double error=0;
  static double errorPrev=0;

  error = abs(velObjetivo) - abs(velMotor);
  pidSum = (Kp * error) + (Kd * (error - errorPrev)) + (Ki * errorSum);
  errorPrev = error;
  errorSum+=error;
  return constrain(motorVal + int(pidSum), 0, 255);
  }
```

Figura 100: Función PID

La función consta de un total de seis variables de entrada:

- "*motorVal*": hace referencia al valor que se le da a los motores para que se muevan. Su rango es por tanto entre 0 y 255.
- *"velObjetivo"*: es la velocidad que queremos que alcance el motor.
- *"velMotor"*: es la velocidad real a la que se mueve el motor.
- *"Kp"*: corresponde con la constante proporcional del controlador PID.
- *"Kd"*: corresponde con la constante derivativa del controlador PID.
- *"Ki"*: corresponde con la constante integral del controlador PID.

Definimos igualmente distintas variables que serán usadas para los cálculos y las inicializamos a 0.

En primer lugar tenemos que ver cuál es nuestro error. Para ello declaramos dicha variable y la definimos como la diferencia entre la velocidad que queremos alcanzar y la velocidad real de nuestro motor.

A continuación pasaríamos a definir la variable "*pidSum*" cuya suma consiste en los tres elementos del regulador PID.

- Primero la parte proporcional con la variable "*kp*" que multiplica al error.
- Después está la parte derivativa. Consiste en multiplicar a la variable "*kd*" por la diferencia de error entre una iteración y la inmediatamente anterior.
- Por último la parte integral en la que multiplicamos a la variable "*ki*" por el sumatorio de los errores de todas las iteraciones.

Seguidamente establecemos los nuevos valores para el error previo y para el error acumulativo y finalmente pedimos a la función que nos devuelva el nuevo valor que le correspondería al motor para alcanzar la velocidad deseada. Esto se consigue sumando o restando al valor actual del motor por el regulador PID.

5.3.2.3.2. Función de corrección de velocidad

Tras haber definido la función PID que nos permitiese modificar la velocidad para alcanzar un valor concreto, se diseñó otra enfocada con la corrección de la velocidad. Se pretendía diseñar una función que fuese capaz de mantener una velocidad constante para el movimiento de los motores *(Figura 101)*.

Para ello debíamos establecer una velocidad de movimiento del motor, medir dicha velocidad y aplicar la corrección del error mediante el PID para posteriormente obtener un nuevo valor que se aplicaría de nuevo al motor. De nuevo mediríamos la velocidad, corregiríamos y así sucesivamente.

```
int correcVel(int motorI, int veloc){
  unsigned long tiempo1 = millis();
  contadorI();
  if(tiempo1 >= tiempo2 + 500){
    velMotorL = (1250*(double)(contadorL-contadorLprev))/(double)(tiempo1-tiempo2);

    contadorLprev = contadorL;
    tiempo2 = tiempo1;
    Serial.println(velMotorL);
    int corrI=PID(motorI,veloc,velMotorL,3,1,3);
    setMotorVal(corrI,0);
    motorI = corrI;
  }
}
```

Figura 101: Función corrección de velocidad

Nuestra función consta de dos variables de entrada:

- "*motorI*": define el valor inicial que queremos darle al motor.
- "*veloc*": este valor indica la velocidad que queremos alcanzar.

En primer lugar llamamos a la función del contador para poder calcular la velocidad del motor cada cierto tiempo en intervalos iguales (esta parte se ha sacado de la función de velocidad).

A continuación y tras establecer los nuevos valores de tiempo y contador previos para sucesivas iteraciones, aplicamos el corrector PID. En él establecemos las variables de entrada como dato así como la velocidad real del motor calculado unas líneas antes. El valor de las constantes se obtiene tras sucesivos ensayos buscando minimizar el error (estos tres valores serán distintos en función del valor que queramos aproximar).

Finalmente utilizamos el nuevo valor proporcionado por el PID para aplicarlo al motor y establecemos ese valor como nueva variable de entrada para la siguiente iteración.

Tras distintas pruebas se decidió prescindir de estas funciones puesto que la resolución de los enconders de nuestro robot es muy baja (son un total de 48 pulsos por vuelta). Esta baja resolución es lo que creaba problemas a la hora de que el robot funcionase.

La velocidad nunca acababa por ajustarse bien. El objetivo era que los motores oscilasen buscando el valor de velocidad deseado para que al cabo del tiempo se estableciesen en un valor constante; en nuestro caso no era así, nunca conseguíamos alcanzar ese valor constante. Se nos planteaban dos situaciones:

- Si las medidas de tiempo eran muy grandes, obteníamos unos valores de velocidad muy fiables y estables. El problema se hallaba al aplicarle el PID ya que al darle un nuevo valor al motor y volver a medir se producía una descompensación entre el movimiento de las ruedas real y el cambio realizado en el programa.
- Por otro lado si las medidas de tiempo eran muy pequeñas para tener así una mayor velocidad de respuesta, lo que sucedía en este caso era que los valores de velocidad que obteníamos variaban mucho entre sí. Perdíamos precisión, ya que dos iteraciones sucesivas de velocidad nos daban resultados distintos de velocidad manteniendo constante el valor del motor. Por lo tanto al aplicar el PID lo hacíamos con velocidades dispares haciendo inútil su corrección.

5.3.3. Función Setup()

Una vez hemos realizado todas las funciones principales que serán usadas para el funcionamiento del robot, es el momento de seguir con la estructura principal.

El siguiente punto a realizar es construir la función "*Setup()*". Como hemos indicado anteriormente esta función se produce una sola vez a lo largo de toda la programación.

Cuando se ejecute el programa y el robot comience a funcionar, el código presente en esta función se lee una única vez y estará operativo (si así se desea) durante todo el tiempo en el que el robot esté funcionando.

Veamos esta función completa y expliquemos cada una de las partes *(Figura 102)*.

```
void setup() {
  // Set pin directions
  pinMode(MUX_A_PIN, OUTPUT);
  pinMode(MUX_B_PIN, OUTPUT);
  pinMode(MUX_C_PIN, OUTPUT);
  pinMode(MUX_D_PIN, OUTPUT);

  pinMode(L_DIR_PIN, OUTPUT);
  pinMode(L_VEL_PIN, OUTPUT);
  pinMode(R_DIR_PIN, OUTPUT);
  pinMode(R_VEL_PIN, OUTPUT);

  pinMode(LED_R_PIN, OUTPUT);
  pinMode(LED_G_PIN, OUTPUT);
  pinMode(LED_B_PIN, OUTPUT);
  pinMode(LED_IR_PIN, OUTPUT);

  pinMode(SERVO_PIN, OUTPUT);

  pinMode(PUSH_PIN, INPUT);
  // Enable pullup resistor for pushbutton pin
  digitalWrite(PUSH_PIN, HIGH);

  // Configure servo scanner
  servoIR.attach(SERVO_PIN);

  // Stop Motors (Otherwise pin is left floating and robot will randomly move
  setMotorVal(0,0);

  // Enable serial communication
  Serial.begin(57600);
  Serial.println("Hello World");

  // Blink the LED in white to say hello

  digitalWrite(LED_B_PIN , HIGH);
  delay(250);
  digitalWrite(LED_B_PIN , LOW);

}
```

Figura 102: Función Setup() 1

Como se puede apreciar en la imagen en primer lugar establecemos las direcciones de los pines del robot. Tanto entradas como salidas. Aquí se encuentran todos los pines que aparecen durante el programa: multiplexor, motores, leds, servo.

También se habilita el pin del botón push y se establece para un valor alto. Este botón sirve para reiniciar la programación del robot una vez que éste conectado.

En esta parte también hemos querido incluir la función de los motores y establecer ambos valores a cero. Esto se hace como medida de seguridad, así nos aseguramos que en primer lugar los motores siempre estarán parados.

A continuación establecemos la velocidad de transmisión (tasa de baudios) a 57600 y mandamos una impresión para comprobar que la comunicación se produce. Debemos recordar que para poder ver toda la información en la pantalla de Arduino, es necesario configurar la aplicación del Monitor Serial a la misma velocidad de transmisión que hemos establecido en el programa.

Finalmente para comprobar que todo funciona y que el robot está conectado establecemos un parpadeo led indicando que el robot se encuentra encendido.

Esta es la parte genérica y principal del programa "*Setup()*". Sin embargo a parte de establecer aquí todos esos comandos nombrados anteriormente, también colocamos en este sitio todas aquellas funciones que hemos diseñado para el programa principal del robot y que solo deben ejecutarse una vez. La posición de todas estas líneas de código están situadas a continuación de lo anterior *(Figura 103)*.

```
servoIR.write(90);

////mapeoInicial(20);

escaneoInicial(2,6);
delay(500);
direccionInicial();
acercamientoPared();

referencia();
//refer = 15;
servoIR.write(0);
aTiempo = 0;
do{
  leerSensores();
  parpadeo(0,1,0);
}while(USD == USI && USD == 16);
```

Figura 103: Función Setup() 2

Como se ve en la imagen se han incluido alguna de las funciones. En primer lugar establecemos el servo en posición de 90° para que el sensor IR mire al frente. A continuación realizamos el escaneo inicial de la habitación y establecemos el número de barridos y la amplitud entre medidas. Tras esto realizamos las operaciones para que calcule la dirección más conveniente y hacemos que el robot se dirija allí.

Finalmente y con el robot colocado incluimos la función que tome el valor de referencia de distancia con la pared. También existe la opción de indicarle al robot que valor de referencia queremos que tome en el caso de preferir alguno en concreto.

Del mismo modo también establecemos el valor inicial del contador de tiempo en ese momento para usarlo posteriormente en la creación de los mapas.

Como último detalle hemos incluido al final un pequeño bucle "*do*" que activa los sensores y toma medidas de ellos. La condición para salir es que los sensores de ultrasonidos tengan un valor distinto de 16. Esto lo hacemos así porque hemos podido comprobar tras múltiples ensayos que en ocasiones estos sensores marcan al principio esas distancias independientemente de la distancia de los objetos que tengan delante. Con esta medida garantizamos que los sensores se estabilicen para que a partir de ese momento den los valores correctos.

En el caso de realizar otro programa distinto que requiera de otras especificaciones modificaríamos en mayor o menor medida esta parte. Es por ello que aquí también podemos colocar al principio de estas líneas de código la función de mapeado inicial como alternativa al mapeado según avanza el robot.

5.3.4. Función Loop()

Esta es la función que se ejecutará continuamente en el programa una vez el robot esté conectado y haya realizado la función "*Setup()*". Aquí es donde se introducen todos los programas e instrucciones que queremos que realice nuestro robot.

Las posibilidades son enormes y en base de lo que se encuentre escrito el robot seguirá unos comandos u otros en función de la situación que se le presente.

A continuación vamos a mostrar los comandos necesarios para el mapeado de una habitación a medida que el robot se desplaza *(Figura 104)*. Aquí incluimos las funciones de los distintos casos con los que se puede encontrar el robot en su avance.

```
void loop()
{

  setMotorVal(120,111);
  contadorI();
  tiempoRef  = millis();
  if(tiempoRef >= tiempoEscaneo + 3000){
    tiempoEscaneo = tiempoRef;
    IntroducirDato = 1;
    if (alfa == 0){
      incrAvance = incrAvance + 42 * 3;
    }
  }
  leerSensores();

  caso1();
  caso2();
  caso3();

  matrizDatos();
  IntroducirDato = 0;
}
```

Figura 104: Función Loop()

En primer lugar conectamos los motores y el contador de los enconders. Así mismo inicializamos temporizador.

Una vez tenemos todos los parámetros es el momento de ver las condiciones de funcionamiento. Lo primero con lo que nos encontramos es con una condición de tiempo. El programa realizará continuamente comprobaciones de tres segundos (este valor puede variar) con el objetivo de tomar medidas con esa frecuencia de tiempo independientemente de si el robot ha girado o no.

Debido a la limitación del robot y a la cantidad de datos que se pueden almacenar en él, hemos decidido utilizar como condición que el programa evalúe la posición del robot con una frecuencia de tres segundos.

Si nos encontramos dentro del bucle lo que hace el programa es reiniciar los valores de tiempo para la próxima vez que pase ese intervalo de tiempo. Asimismo activamos la variable de "*IntroducirDato*" para indicar que estamos en un tramo recto y que queremos tomar las medidas en ese instante.

Ya hemos dicho que cuando el robot avanza en línea recta el ángulo alfa tiene un valor de 0. Si es así entonces ha sufrido un incremento de avance. Es por ello que cuando esto ocurre lo que hacemos en él es aumentar el valor de "*incrAvance*" en un valor fijo de 42 contadores cada vez que se accede a este bucle. Este valor ha sido estimado con la distancia aproximada que avanza el robot cada vez que entramos en esta función. Como las medidas que tomamos son cada tres segundos, el valor del incremento será el triple.

Tras esto finalizamos la condición temporal para los tramos rectos. A continuación lo que nos encontramos es la función que toma las medidas de los sensores. Esto lo hace en cada ciclo, pero no todos los valores se guardarán.

Como el robot va a estar en continuo movimiento por una zona, tras leer el valor proporcionado por los sensores hará que en función de los datos obtenidos entremos en un caso u otro.

Con los datos proporcionados por los distintos casos el programa irá poco a poco creando las distintas matrices con los datos del mapa de la zona.

En el caso de encontrarnos en el "*caso1()*" el robot continua recto, por lo que los datos de las matrices serán una continuación directa en dirección y sentido que el valor inmediatamente anterior, teniendo en cuenta la estimación de avance del robot cada tres segundos.

Si por el contrario nos encontramos ante el "caso2()" o "caso3()" el valor de las matrices indicarán el giro del robot y la nueva dirección tomada.

Tras evaluar cada uno de los casos, el programa entra en la función que guarda las posiciones del robot. En función de estos valores se decidirá si deben ser almacenados. Es por ello que tenemos varias posibilidades con estos datos:

- Las medidas obtenidas **NO** forman parte de la condición de tiempo y **NO** son críticas, por lo que no intervienen en el giro del robot. En este caso los valores se descartan.
- Las medidas obtenidas **SÍ** forman parte de la condición de tiempo y **NO** son críticas, por lo que no intervienen en el giro del robot. En este caso los valores se guardan debido a que forman parte de una medida intermedia en un tramo recto.
- Las medidas obtenidas **NO** forman parte de la condición de tiempo y **SI** son críticas, por lo que intervienen en el giro del robot. En este caso los valores se guardan al tratarse de un giro, y por tanto de un punto clave.
- Las medidas obtenidas **SI** forman parte de la condición de tiempo y **SI** son críticas. Es difícil que se produzca esta situación concreta, pero en el caso de que así sea tiene más importancia la condición de giro, por lo que guardamos este valor como tal.

De esta manera conseguimos con estas funciones crear las matrices necesarias para la creación de los diferentes mapas basados en la medida de los sensores. La creación de estos mapas termina tan pronto como desconectemos el robot.

Como última línea indicamos al programa que debe reiniciar el valor de "*IntroducirDato*" en el caso de que se hubiese activado.

Del mismo modo y como una alternativa más simple, hemos creado otros dos programas distintos usando diversas funciones declaradas al principio y que nos permite también mostrar las diferentes características del robot.

Una de las ventajas que tenemos ahora es que los programas que incluyamos pueden ser muy simples, ya que al declarar las funciones anteriormente aquí solo es necesario llamarlas para que cumplan su función.

5.3.4.1. Programa 1

Este programa quiere mostrar el funcionamiento de los motores; así como del servo y del sensor infrarrojo. Veamos en que consiste *(Figura 105)*.

En primer lugar debemos llamar a las funciones de los contadores y de las velocidades de los motores. Esto es básico para el funcionamiento del robot.

A continuación se conecta el servo y se habilita el sensor de infrarrojo. De este modo realiza un escaneado de la zona con una amplitud igual a la capacidad máxima del servo (180°).

```
void loop()
{

  /////PROGRAMA 1/////


  contadorI();
  contadorD();
  velMotorI();
  velMotorD();

  Mservo(1);
  giro(120,'L');
  delay(5000);

  datos();
}
```

Figura 105: Programa 1

En cada una de las posiciones del servo, el sensor infrarrojo actúa tomando medidas de los distintos obstáculos que se encuentren y mostrando la distancia a la que se encuentran.

Cuando el servo termina de escanear una parte, las ruedas del motor girar una amplitud de 120° para así poder ver otra zona.

Con esto el proceso vuelve a repetirse, con lo que el servo y el sensor volverán a escanear esta nueva zona y reflejarán los datos en la pantalla de Arduino.

De este modo conseguimos que el robot dé una vuelta completa y escanee un recinto con una amplitud de 360°.

5.3.4.2. Programa 2

El segundo programa está destinado al desplazamiento del robot a través de un entorno. Mediante los distintos sensores el robot debe decidir hacia dónde dirigirse *(Figura 106)*. Seguidamente explicaremos el código.

En primer lugar llamamos a las funciones correspondientes a los tres sensores. Los dos de ultrasonidos y el de infrarrojo. En esta ocasión el servo se encuentra desconectado. De esta forma tenemos al sensor de infrarrojo tomando las medidas de los obstáculos que se encuentren enfrente y a los sensores de ultrasonidos enfocados a ambos lados del robot.

A continuación y al igual que en el programa anterior declaramos las funciones correspondientes a los contadores y velocidades de ambos motores.

Para que el robot pueda desplazarse hacemos un llamamiento a la función de desplazamiento y establecemos uno de los modos. Como el desplazamiento será indefinido es necesario ponerlo en los datos de entrada.

```
void loop()
{

  /////PROGRAMA 2/////

  int USIzdo = readUltrasonicSensor('L');
  delay(100);
  int USDcho = readUltrasonicSensor('R');
  delay(100);
  int SenIR = readIRSensor();
  delay(100);

  contadorI();
  contadorD();
  velMotorI();
  velMotorD();

  avance(2,0);
  if (SenIR < 10){
    giro(180,'L');
    parpadeo (1,0,0);
  }

  if(USIzdo <10){
    if (USIzdo >3){
      giro(90,'R');
    }
  }

  if(USDcho <10){
    if (USDcho >3){
      giro(90,'L');
    }
  }

  Serial.print("Ultrasonidos Izdo: "); Serial.println(USIzdo);
  Serial.print("Ultrasonidos Dcho: "); Serial.println(USDcho);
  Serial.print("Infrarrojos: "); Serial.println(SenIR);

}
```

Figura 106: Programa 2

Una vez establecidas las funciones principales del programa, lo siguiente es elegir las condiciones de movimiento del robot en función de los sensores. Como condiciones de desplazamiento hemos establecido que:

- Si el sensor de infrarrojos detecta un objeto en una distancia inferior a los diez centímetros, el robot dará media vuelta al encontrarse con un obstáculo frontal. En este caso el led RBG parpadeará durante el giro del robot.
- Si el sensor de ultrasonidos izquierdo detecta un objeto a una distancia entre tres y diez centímetros, el robot girará 90° hacia la derecha.
- Si el sensor de ultrasonidos derecho detecta un objeto a una distancia entre tres y diez centímetros, el robot girará 90° hacia la izquierda.

Para evitar errores con falsas lecturas de los sensores de ultrasonidos hemos establecido una distancia mínima de tres centímetros, ya que a distancias muy cortas este sensor no detecta bien los objetos. Es necesario un margen.

6. Desarrollo y construcción

6.1. Planteamiento

El robot consta como hemos dicho, de tres partes diferenciadas: electrónica, mecánica e informática. En los apartados anteriores hemos visto en que consistían cada uno de ellos y que elementos han sido necesarios para su elaboración.

En este apartado queremos tratar a todas estas partes en conjunto, indicando como hemos pasado de una fase a otra y como se ha ido montando el robot.

Del mismo modo que tenemos tres partes diferentes en este desarrollo, también dividiremos en esas tres mismas fases el proceso por el cual se ha ido construyendo el proyecto.

- Creación de la placa.
- Construcción y montaje del chasis.
- Diseño de la programación.

En primer lugar y antes de empezar a hacer nada, es muy importante tener fijado el objetivo que queremos alcanzar con este proyecto.

Una vez tenemos claros los conceptos y la idea a seguir para desarrollarlo, es momento de empezar a diseñar el robot.

6.2. Creación de la placa

El elemento central del robot es la placa o PCB. Este elemento será el que sustente y fije al resto de componentes.

Una vez tenemos las dimensiones y la forma que tomará la placa para su diseño final, es el momento de crear un circuito acorte al tamaño de nuestra placa y que reúna todas las características necesarias para que el robot pueda realizar todas aquellas tareas que queramos programarle.

En esta parte es importante saber que sensores y demás componentes debe llevar el robot. El espacio el limitado y debemos elegir correctamente que incluir y que no.

Otro factor importante en la elección de los componentes es el económico. No debemos olvidar que queremos crear un robot de bajo coste y esto puede limitar en cierta medida los elementos a incluir.

El diseño del circuito ha sido realizado con la ayuda de programas informáticos que facilitan esta tarea (KiCad).

Con el circuito creado es el momento de diseñar la placa de tal forma que cumpla con las especificaciones de tamaño y sea capaz de contener a todo el circuito.

Tenemos que tener en cuenta que la placa tiene varios orificios que corresponden con:

- Hueco de las dos ruedas.
- Cuatro tornillos de enganche al chasis.
- Diversas perforaciones en todo el perímetro de la placa para futuras ampliaciones.

Mediante el mismo programa disponemos de la disposición de los canales y de las perforaciones de las conexiones para la comunicación y conexión de todos los componentes que vayan a estar incluidos en la placa.

Una vez diseñada la placa es el momento de fabricarla. Para ello hemos enviado los archivos al departamento de automática de la universidad para que nos la construyan de acuerdo con las especificaciones deseadas.

Ya hemos indicado que para la creación de nuestra PCB no se había realizado las fases de serigrafía y máscara antisoldante.

La placa ha sido diseñada con orificios pasantes; esto significa que el montaje de todos los componentes electrónicos se realiza pasando las patillas de cada uno de ellos por sus orificios correspondientes en la placa de tal modo que la atraviesen y posteriormente soldarlos en la parte inferior.

En la siguiente imagen podemos comprobar cómo se ha realizado el soldado final de todas las piezas *(Figura 107)*.

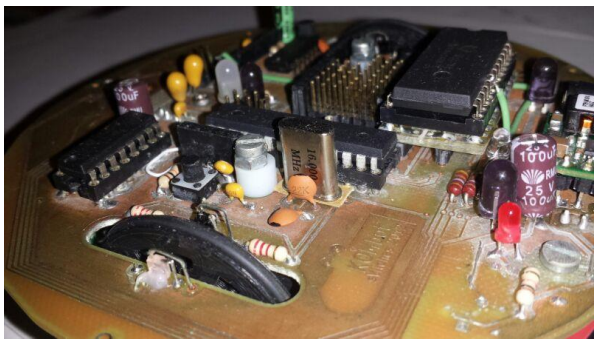


Figura 107: Soldadura de la placa

Como detalle mencionamos la colocación de los encoders de las ruedas. Una vez soldadas, éstas deben doblar sus patillas de tal modo que el cabezal del encoder se sitúe apoyado en la superficie de la placa y en la zona central del hueco de la rueda. Es importante una buena alineación para que así pueda recoger la información de las ruedas correctamente llegado el momento.

El servo del motor se sitúa en la placa mediante dos tornillos que aseguran su posición. Tras su colocación deberemos enchufarlo en su conector correspondiente.

También es remarcable indicar que en esta parte no conectamos el sensor de infrarrojos debido a que no va soldado a la placa; sino acoplado al servo mediante una torreta. Lo colocaremos en el siguiente apartado, cuando ya esté el chasis montado.

Debido a la no inclusión de la máscara antisoldante hemos tenido diversos problemas de oxidación de las vías de algunos de los componentes. En ocasiones esto provocaba que varias vías que tendrían que estar separadas se mantuviesen unidas como si formasen parte de la misma. En futuras ampliaciones deberemos corregirlo.

Como consecuencia algunos elementos del circuito no funcionaban o lo hacían incorrectamente dando lugar a errores. Este problema se ha repetido en varias partes del circuito, obligándonos a repasar cada una de ellas para intentar solventarlo. Para solucionarlo y debido a que esas pistas estaban dañadas, hemos tenido que colocar algunos cables *(Figura 108)* para que unan esas partes de tal modo que la corriente pase por ellos y eviten su recorrido por la zona afectada.

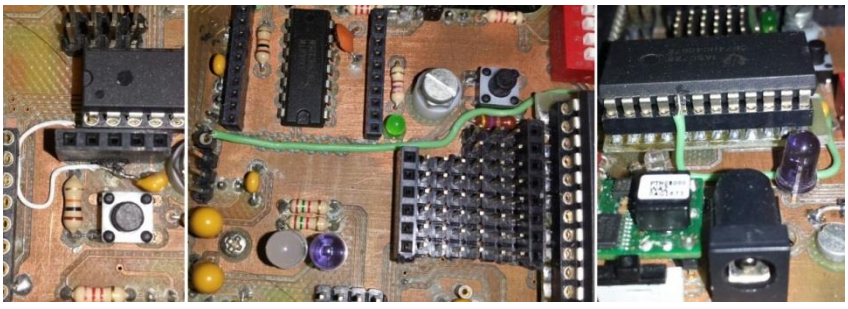


Figura 108: Cableado del circuito

6.3. Construcción y montaje del chasis

Una vez tenemos construida y soldada la placa, es el momento de centrarse en el chasis. El diseño del chasis está basado en las necesidades de la PCB, hemos tenido que tener en cuenta ciertos factores como:

- Las dimensiones de la placa.
- Los orificios para las ruedas.
- La altura de colocación de los motores para la colocación de las ruedas.
- La posición de la batería.

El diseño de las ruedas también está pensado para que su movimiento sea captado por los encoders.

Una vez tenemos claro que diseño hacer, es el momento de crearlo. Para ello lo primero es diseñar el modelo del chasis en el ordenador para después poder imprimirlo físicamente.

Tras una serie de versiones distintas y gracias al uso de programas informáticos como Qt Creator y OpenScad tenemos por fin el diseño final del chasis.

Una vez tenemos el diseño final del chasis y de las ruedas es el momento de imprimirlos. Usamos para ello las impresoras 3D de la universidad y los programas Replicatorg o Pronterface & Slic3r.

Tras unos 45 minutos para el chasis y 10 minutos para cada rueda, tendremos listas nuestras piezas y listas para ser montadas. Este tiempo es orientativo; ya que puede variar algo en función de la impresora y configuración elegida.

Para realizar el correcto montaje del chasis en la placa *(Figura 109)* deberemos colocar en primer lugar la batería. La introduciremos en el hueco destinado a ella en la posición que indica la figura, asegurándonos de que los cables se sitúan en el lado del chasis que tiene la cavidad destinada a ellos.

La batería se apoya en el puente del chasis y, en esta posición debería estar lo suficiente introducido como para que se quede al mismo nivel de superficie que el resto del chasis.

El siguiente paso consiste en atornillar el chasis a la placa. Tenemos un total de 4 orificios (dos en la zona de la batería y otros dos en el hueco de los motores). Estos orificios coinciden con los que se encuentran en la PCB; tan solo deberemos hacerlos coincidir y sujetarlos con tornillos y arandelas.

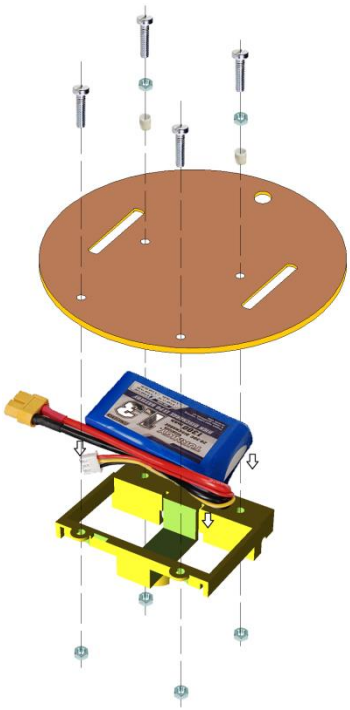


Figura 109: Montaje de chasis y batería

Una vez acoplado el chasis, es el momento de colocar los motores de las ruedas. Antes de introducirlos en su cavidad, deberemos colocar los ejes de montaje de las ruedas en el eje de giro de motor.

Tras asegurarlo, introducimos los motores en las cavidades del chasis para las que están destinadas. Los motores se introducen deslizándolos hasta que queden fijos.

Paralelamente introducimos en las ruedas de plástico las gomas para evitar el deslizamiento de las mismas cuando éstas comiencen a rodar. Su colocación es sencilla, tan solo habrá que estirar un poco la goma para que podamos introducirla en el canal de la rueda.

Posteriormente unimos la rueda al eje de montaje. Mediante cuatro tornillos conseguiremos fijarlas. Para poder colocarlo correctamente deberemos posicionar primero la rueda en el hueco de la placa destinado a ello; una vez colocada en este espacio lo alineamos con el eje de montaje y atornillamos.

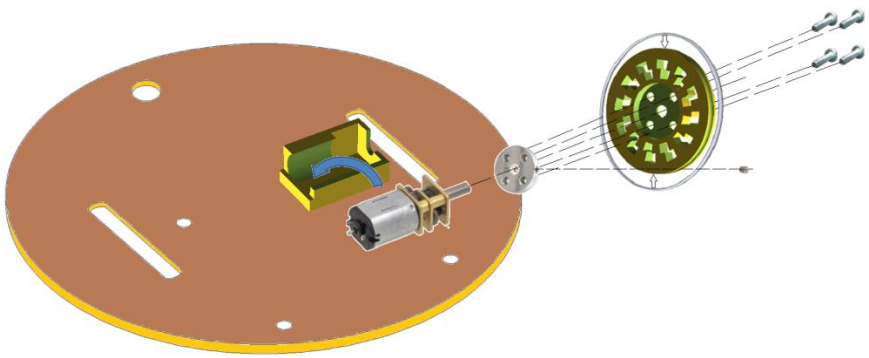


Figura 110: Montaje de la rueda

Tras colocar las ruedas colocaremos las canicas en sus cavidades para que actúen como ruedas locas y equilibren al robot. Haciendo un poco de presión podremos introducirlas en sus huecos. Esto se puede apreciar en la imagen anterior *(Figura 110).*

Cuando los motores y la batería se encuentren colocados en su posición, podemos conectarlos a la parte inferior de la placa mediante sus respectivos conectores.

Con todo el chasis montado en la placa solo nos queda colocar el sensor de infrarrojo al servo.

En nuestro caso hemos unido el sensor a un pequeño listón de madera a modo de torreta. De este modo permitimos al servo poder girar libremente sin que le estorbe la posición del sensor.

Para poder colocar esta “torreta” tenemos que unirla al servo mediante un adaptador. Usamos para ello una de las hélices que venían incluidas como accesorio en el set del mini servo.

Para impedir que los extremos de las hélices choquen con el resto de componentes del robot cuando el servo gire, será necesario cortar las aspas; de este modo nos quedaremos únicamente con la zona central. De este modo pegamos uno de los extremos a la “torreta” y el otro se coloca en el servo.

Una vez colocado, solo nos quedará conectar el sensor IR a la placa.

6.4. Diseño de la programación

El diseño del programa corresponde únicamente con la parte de programación. Consiste en crear el programa que el robot ha de seguir en todo momento.

En función de los objetivos que se querían conseguir con este proyecto, estuvimos pensando que movimientos y funciones queríamos que hiciese.

Tras considerar que debe y que no debe hacer el robot, lo único que quedaba por hacer era diseñarlo en Arduino.

Primero se crearon las instrucciones básicas y a partir de ahí se fue desarrollando más hasta tener el programa completo.

Para introducir este programa usamos el adaptador foca y el puerto USB mini.

Este proceso ya ha sido descrito previamente en su apartado correspondiente.

7. Pruebas y funcionamiento

7.1. Introducción

Este apartado recoge todas aquellas pruebas y mediciones que hemos tenido que hacer durante la creación del robot hasta su funcionamiento final.

Veremos tanto las distintas pruebas relacionadas con la electrónica y que nos han llevado a tomar unas decisiones u otras, como las distintas pruebas relacionadas con la programación del robot y en las que hemos comprobado su funcionamiento.

A lo largo de su diseño electrónico hemos tenido que decidir entre varias opciones para elegir los componentes. También se han tenido que realizar distintas mediciones y pruebas que nos permitiesen saber que íbamos por el buen camino.

Así mismo, una vez tenemos nuestro robot montado y programado, es necesario saber si funciona como se espera de él. Por ello comprobaremos todas y cada una de las funciones del robot.

En este apartado no incluimos pruebas de diseño del chasis puesto que para su creación se tuvieron en cuenta las especificaciones de la placa y a raíz de ellas creamos nuestro diseño. Mediante un proceso de prueba y error que nos llevó a diseñar tres versiones del chasis y distintos ajustes en las medidas, dimos con la versión final.

7.2. Pruebas electrónicas

En este apartado nos centraremos en todas aquellas pruebas y cálculos que hemos realizado durante el desarrollo de la parte electrónica.

7.2.1. Protecciones

7.2.1.1. Convertidor DC

Algunas consideraciones que debemos tener en cuenta con respecto al bloqueo de tensión mínima son:

- El UVLO se ha establecido en 6.76 V (es decir 3.38 V por celda). El UVLO se recomienda que sea por lo menos 3.3V de acuerdo con los datos de la siguiente gráfica *(Figura 111)*.

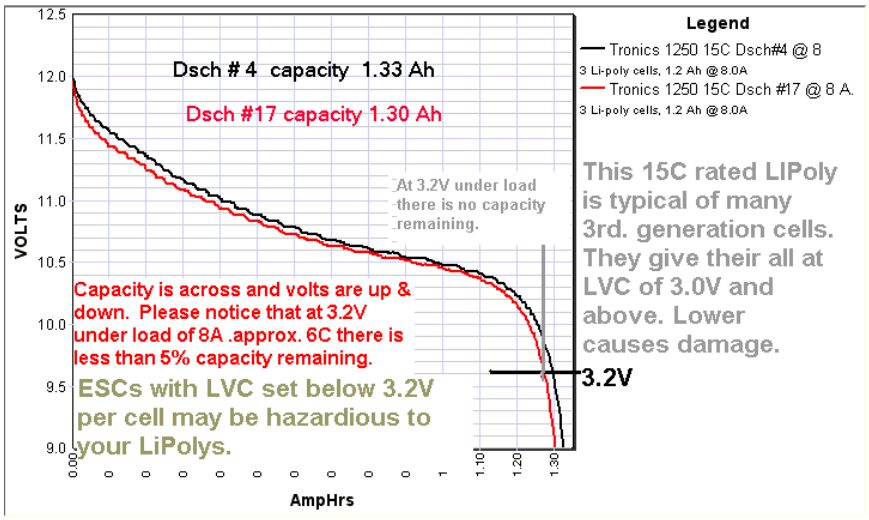


Figura 111: Gráfica UVLO

- El convertidor DC-DC ha sido probado contra cortocircuitos con diferentes cargas (de 0 a 10 ohmios). Ninguna produce suficiente corriente para la activación polyfuse.
- El led UVLO puede activarse/parpadear durante un cortocircuito. En esta situación, el pin 4 del convertidor se establece en aproximadamente 3 V, sin razón aparente. Este comportamiento no es considerado peligroso.

7.2.1.2. Detector de baja tensión

Tal y como dijimos en el apartado 3 del proyecto, la primera opción que se planteó como detector de baja tensión fue un circuito compuesto principalmente por un $V_{Ref\ Zener}$ y un comparador *(Figura 112)*. Es el siguiente:

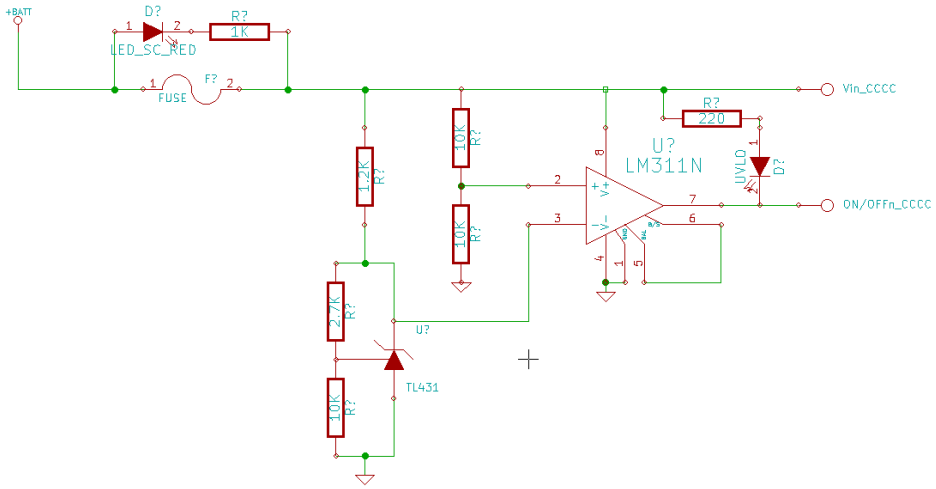


Figura 112: Circuito $V_{Ref\ Zener}$ + Comparador

Durante un cortocircuito a 8.2V → t_{ida} = 150ms, I_{max} = 12 A, I_{ss} = 200mA.

Nota: Este circuito puede comenzar a oscilar. Necesitamos más histéresis.

Teniendo en cuenta que para nuestro caso la V_{ref} ≈ 2.5V, el circuito *(Figura 113)* que le correspondería al regulador de tensión ajustable sería:

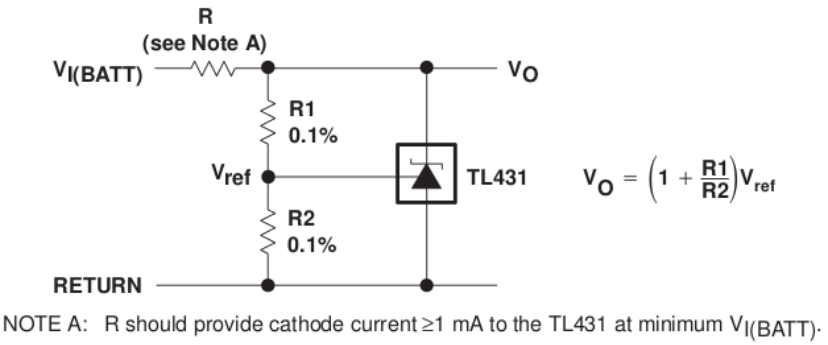


Figura 113: Circuito $V_{Ref\ Zener}$

Del mismo modo que para el componente anterior, el circuito para el comparador *(Figura 114)* con una tensión V_{in} de 12 V y salida de 50 mA es:

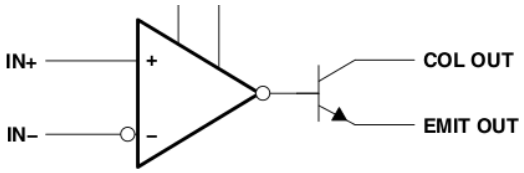


Figura 114: Circuito Comparador

Como ya comentamos anteriormente, existen soluciones más fáciles para solucionar el tema del detector de baja tensión. Es por eso que al final y tras probar el circuito, decidimos descartar esta opción.

7.2.1.3. Fusible

Es importante saber que el fusible elegido (en nuestro caso el PTC reajustable RKEF110) es capaz de soportar los excesos de corriente que pudieran producirse en el robot. Para ello comprobaremos unas mediciones. Éstas han sido realizadas con una sonda de corriente de 0.2 Ω: 1 V = 1 A.

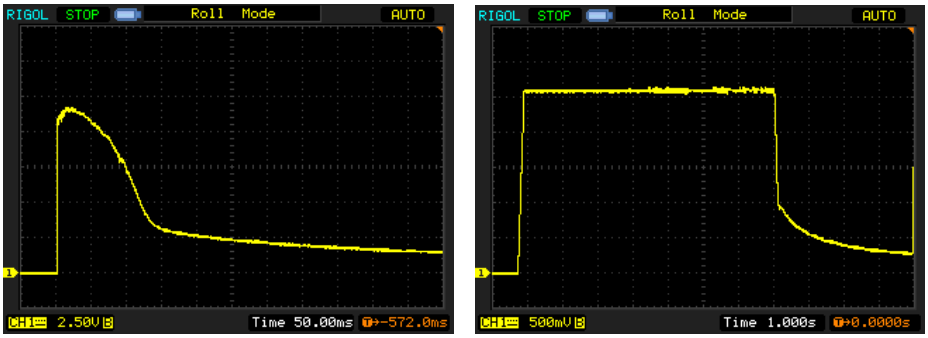


Figura 115: Mediciones del fusible

En la siguiente tabla podemos ver los valores de ambas mediciones.

		t_{ida}	I_{max}	I_{ss}
(Figura 115a)	En cortocircuito a 8.2 V	150ms	12 A	200 mA
(Figura 115b)	A 2.5Amps, 7.4 V	7s	2.5 A	250 mA

7.2.2. Comunicaciones

7.2.2.1. Prueba de XBees

7.2.2.1.1. Placa

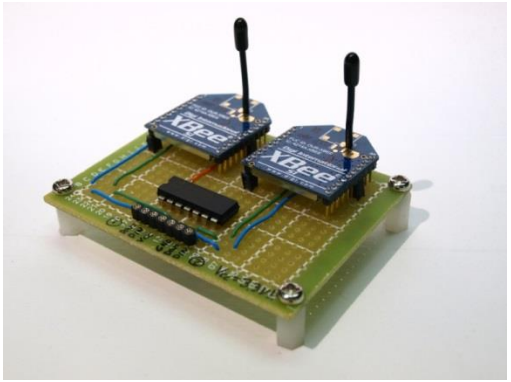


Figura 116: Circuito de prueba de los XBee

Cuando se montó este circuito *(Figura 116)* se quedaron los pines activados del buffer sin conectar. Es importante conectarlos cuanto antes; sin embargo parece que funciona sin ellos porque podría tener resistencias internas.

Un elemento importante de este circuito corresponde con el buffer de 3 estados *(Figura 117).*

❖ ***Buffer 3 estados cuádruple 74ABt125***

• Número de pines	14
• Número de canales	4
• Tipo de salida	3 estados
• Tipo de entrada	Terminación única
• Rango de temperaturas	-40 °C y 85 ° C
• Montaje	Orificio pasante
• Tensión de alimentación máxima/mínima	5.5 V / 4.5 V

El pin 14 está reservado a VCC y el pin 7 a GND.

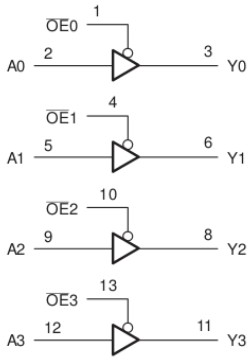


Figura 117: Especificaciones buffer 3 estados cuádruple

A la hora de probarlo se va a utilizar el esquema del kit de adaptador XBee de Adafruit cuyo esquema es el siguiente *(Figura 118)*:

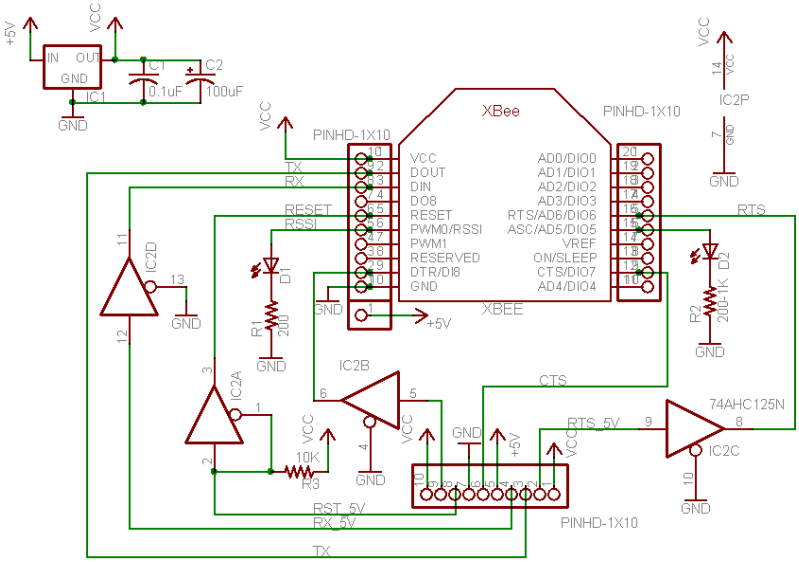


Figura 118: Kit de adaptador XBee de Adafruit

7.2.2.1.2. Conexiones

Para probar los XBee usamos un cable FTDI. El esquema para conectarlo se muestra a continuación *(Figura 119)*:

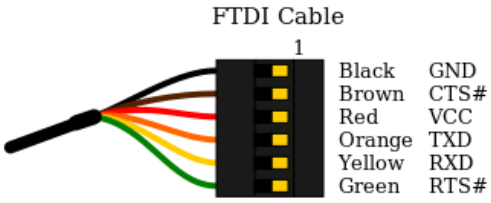


Figura 119: Conexión cable FTDI - XBee

FTDI Pin	XBee Pin
Txd	Din
Rxd	Dout
GND	GND

7.2.2.1.3. Prueba de comunicación

Para probar la comunicación utilizamos el software de Arduino.

Desde la terminal serie de Arduino, ajustado a 9600 baudios, enviamos:

```
+++
```

Es importante asegurarse de que no se está enviando retorno de carro (CR) o nuevos caracteres de línea; ya que de ser así no funcionaría.

Si la comunicación funciona correctamente deberemos recibir:

```
OK
```

Esto ha sido probado con ambas XBees (una en cada posición del tablero). Parece que podría haber un retardo (aprox. 5s) entre dos pruebas "+ + +".

7.2.2.1.4. Configuración de XBees

Para configurar los XBees utilizaremos comandos AT según Ladyada. Vamos a usar minicom, un emulador de terminal serie para Linux. Ejecutarlo por primera vez como root i, a fin de guardar la configuración:

```
sudo minicom -s
```

Desde el menú, seleccionar “configuración del puerto serie” y configurarlo hasta conseguir los siguientes resultados:

```
+-----------------------------------------------------------------------+
| A -    Serial Device              :  /dev/ttyUSB0                     |
| B - Lockfile Location             :  /var/lock                        |
| C -   Callin Program              :                                   |
| D -  Callout Program              :                                   |
| E -    Bps/Par/Bits               :  9600 8N1                         |
| F - Hardware Flow Control  :  No                                      |
| G - Software Flow Control   :  No                                     |
|                                                                       |
|    Change which setting?                                              |
+-----------------------------------------------------------------------+
```

Entonces volveremos con la tecla Escape y guardamos la configuración como dfl.

Volvemos a probar la conexión

Nota: presionando “Crtl+a” y luego “e” a su vez, se activará el eco local (es decir, que puede verse lo que se escribe). Ahora presionamos la tecla + tres veces (sin presionar la tecla Enter).

```
+++
```

Entonces recibiremos:

```
OK
```

Nota: Usando el terminal, podemos cambiar la velocidad de transmisión (tasa de baudios) usando el comando ATBD con un número después que selecciona qué velocidad de transmisión usar:

```
0 = 1200
1 = 2400
2 = 4800
3 = 9600
4 = 19200
5 = 38400
6 = 57600
7 = 115200
```

```
-> + + + (Entrar en modo AT)
<- OK
-> AT (comprobar si el módem de XBee está respondiendo)
<- OK
```

```
-> ATBD (obtener la tasa de baudios de la corriente anterior)
<- 3 (9600)
-> ATBD 4 (establecer la tasa de baudios en 19200)
<- OK
-> ATBD (comprobar otra vez)
<- 4
-> ATWR (escribe la tasa de baudios para cambiar a flash)
<- OK
```

Presionar Enter después de cada comando.

7.2.2.2. Interferencias en los pines Tx/Rx

Cuando se conecta un módulo de bluetooth y el cable de FTDI a los pins Rx y Tx del Atmega, la programación se hace imposible.

Síntomas

- Desconectar del pin etiquetado Rx en Arduino desde Bluetooth lo arregla.
- Desconectar del pin etiquetado Tx en Arduino desde Bluetooth no funciona.
- Desconectar el Bluetooth de VCC lo arregla.

Causa

El módulo Bluetooth tira del pin Tx a 5V (pin etiquetado Rx en Arduino), por lo que es imposible utilizar ese pin para cualquier otra comunicación serie.

Solución

Usar un switch *(Figura 120)* para seleccionar entre el modo de programación y el modo radio:

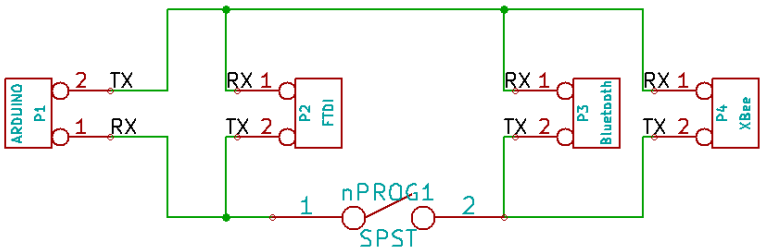


Figura 120: Switch. Modos de programación - radio

En la placa de Arduino, los leds de Tx y Rx están conectados al FTDI GPIOs. Por lo tanto si usamos el cable FTDI no es posible agregar los leds de Tx y Rx, porque podrían cargar esos pins.

7.2.3. Sensores

7.2.3.1. Encoders con circuito inversor

- ***Inversor disparador Schmitt CMOS 14PDIP***

En un principio se pensó utilizar este componente para controlar los encoders; finalmente descartamos esta idea y no se llegó a comprar. Características:

• Número de pines	14
• Tensión de alimentación máxima/mínima	18 V / 3 V
• Montaje	Orificio pasante
• Rango de temperaturas	-55 °C y 125 ° C

La corriente de salida I_{out} es de 0.88 mA para un valor de 0.22*V_{dd} (1V), produciendo histéresis.

Después de probar el diseño con el inversor y observar que el disparador Schmitt y la escala R2R utilizan demasiados componentes y tamaño del tablero, se ha decidido leer los sensores directamente con el ADC, lo que significa:

- Cada vez que queremos leer los encoders de las ruedas tenemos que hacer 4 mediciones de ADC (@ = 125 µs cada 500 µs).
- Es fácil añadir una rutina de calibración.

Cada rueda tiene dos pares de fototransistores-led. Por lo tanto vamos a tener que medir cuatro fototransistores *(Figura 121)* con el uController ADC.

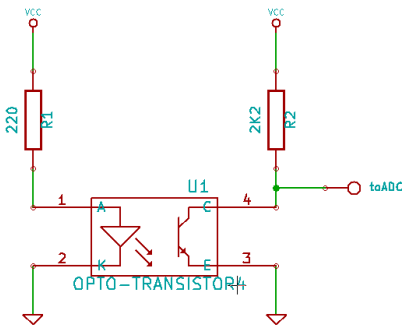


Figura 121: Esquema encoder

La tensión en la salida varía de 0.3 V (cuando el haz de luz pasa a través de la ranura) a 4.4V (cuando el haz de luz está bloqueado).

7.3. Pruebas software

Aquí realizaremos distintas pruebas y mediciones que nos permitirán saber si todas las funcionalidades del robot tienen una buena fiabilidad y realizan su cometido correctamente.

Distinguiremos distintos tipos de ensayos. Por un lado comprobaremos de forma individual cada uno de los sensores y elementos del robot para así ver si tienen una buena precisión; por otro lado realizaremos otro tipo de pruebas enfocadas a movimientos más complejos que incluyen una combinación de varios de los sensores de los que disponemos funcionando al mismo tiempo.

7.3.1. Funciones básicas

En primer lugar veremos por separado cada uno de los sensores y elementos de los que consta el robot. Tenemos que comprobar si realmente tienen una buena precisión y si responden correctamente.

7.3.1.1. Sensores frontales

Los sensores frontales son los encargados entre otras cosas de detectar los objetos y obstáculos que pueda encontrarse el robot en su recorrido. Es muy importante que estén bien calibrados y que sean capaces de detectar los objetos a tiempo para que así el robot actué en consecuencia y pueda esquivarlo si fuese necesario.

Estos sensores también pueden utilizarse para medir las distancias a las que se encuentran los objetos; de este modo podemos hacernos una idea de cómo está distribuida una zona y que elementos hay en ella. Debemos tener una gran precisión para saber con fiabilidad la situación de cada uno de estos elementos.

Distinguimos frontalmente tres sensores distintos. Por un lado tenemos un sensor de infrarrojo situado en la parte central encima del servo. Es el sensor que se encuentra más elevado en el robot. La ventaja de estar colocado encima del servo es que podemos hacer girar éste para que el sensor de infrarrojo pueda captar los objetos que pueda haber en un cierto ángulo o bien dejarlo fijo para que siempre mida un punto en concreto.

Los otros dos sensores que encontramos son de ultrasonidos y se sitúan a ambos lados del sensor de infrarrojo, en la parte inferior del robot. Estos sensores permiten captar los elementos que pueda haber en los extremos. Al contrario que el anterior, estos son fijos. Se encargan de medir las distancias de los objetos que puedan aparecer por los lados para así poder evitarlos.

Para probar la precisión de nuestros sensores procederemos a colocar al robot en una posición fija y pondremos debajo del sensor a estudiar una cinta métrica que nos indique la distancia.

Colocaremos en frente de cada sensor un objeto a distintas distancias y posteriormente encenderemos el robot para que el sensor vaya captándolos y nos muestre por pantalla la distancia que capta. Finalmente compararemos los datos que recibimos del sensor del robot con los datos reales a los que hemos situado el objeto y compararemos de esta forma que precisión tiene.

Situaremos en objeto en primer lugar en un punto muy cercano al robot y poco a poco iremos alejándolo. De esta forma podremos saber que amplitud de mediciones tenemos.

7.3.1.1.1. Sensor frontal de infrarrojo

En primer lugar veremos la precisión del sensor de infrarrojo. Le dejamos en la posición que tiene por defecto, de tal forma que mire al frente.

Según la hoja de datos del fabricante, este sensor está pensado para medir distancias entre 10 y 80 cm. Realicemos pues las mediciones de este sensor.

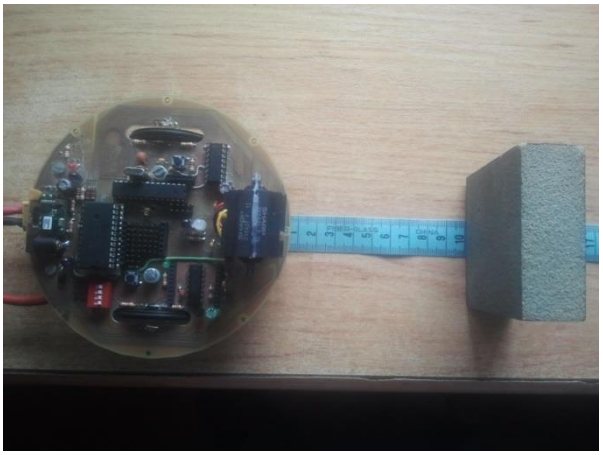


Figura 122: Medida sensor infrarrojo

En la imagen *(Figura 122)* podemos ver el robot frente al objeto que hemos usado para medir las distancias. Iremos separando la distancia del obstáculo y tomando distintas medidas.

Los datos que muestra el sensor se corresponden con las medidas a las que se encuentra el objeto en tiempo real. Seleccionando los datos obtenidos con las posiciones reales a las que hemos colocado el objeto podemos ver en la siguiente tabla comparativa el rango de medidas para su posición real y el valor obtenido para esa misma distancia (los resultados están expresados en cm).

Real	10	20	30	40	50	60	70	80
Sensor	10	20	31	42	52	63	73	82

Como podemos ver en la tabla los resultados se aproximan bastante al valor real. Sin embargo debemos destacar un par de cosas:

- Este sensor basa su resultado en una ecuación potencial cuyo valor viene dado por diferencias de tensión. Esto hace que en ocasiones los resultados obtenidos para objetos que se mantienen a una misma distancia no den como resultado valores absolutos, si no que oscilen entre una medida y otra.
- A medida que los objetos se sitúan más lejos, la precisión del sensor disminuye, haciendo que el resultado obtenido sea distinto a la distancia real.
- Estas variaciones de distancia no suponían una gran diferencia con el valor real; diferían entre 1 y 3 cm. En distancias mayores las variaciones eran mayores (entre 3 y 7 cm).
- Este tipo de sensor es sensible a efectos como la luz o el clima, proporcionando un resultado erróneo. Esto también ocurre con objetos cuya superficie sea reflexiva.

7.3.1.1.2. Sensor de ultrasonidos izquierdo

Una vez visto el sensor central, procederemos a estudiar los sensores de ultrasonidos. Éstos se comportan de forma distinta al anterior a la hora de captar los objetos. Hemos tenido que programar este sensor teniendo en cuenta la velocidad de desplazamiento del sonido; ya que depende de ella.

Según la hoja de datos el rango que estos sensores son capaces de captar se encuentra entre 0 y 150 cm.

De la misma forma que antes situaremos el objeto delante del sensor a distintas distancias y anotaremos el resultado que nos muestra por el monitor *(Figura 123)*.

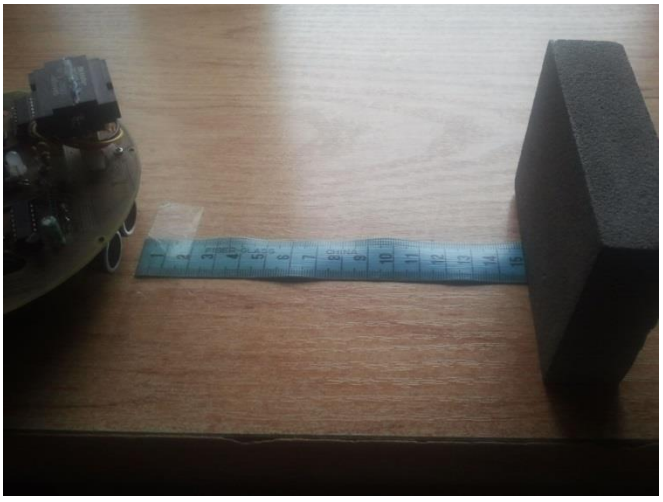


Figura 123: Medida sensor ultrasonidos

Volvemos a seleccionar los datos y mostramos los resultados (en cm) en la siguiente tabla comparativa:

Real	5	10	20	30	40	50	75	100	125	150
Sensor	5	10	20	30	39	49	74	100	124	148

Podemos comprobar que los resultados se aproximan bastante a la realidad. En este caso se ve que el error que nos marca el sensor es de 1 o 2 cm.

Sin embargo cabe destacar que en ocasiones el sensor daba como resultado una distancia errónea que se alejaba bastante de la que realmente se encontraba el obstáculo. Todo depende de cómo rebote el eco antes de volver al sensor.

Este es un error frecuente que se produce en este tipo de sensores; ya que como inconveniente tienen que producen falsas alarmas. También tienen el problema de existir una zona ciega en la cual no se puede detectar el objeto. Los resultados óptimos se obtienen cuando el obstáculo se encuentra perpendicular al sensor.

7.3.1.1.3. Sensor de ultrasonidos derecho

En este caso tenemos la misma situación que el sensor izquierdo. Realizaremos las medidas de igual forma que antes y compararemos resultados. Puesto que ambos sensores corresponden al mismo modelo, los resultados deberán ser similares.

La tabla en este caso nos queda:

Real	5	10	20	30	40	50	75	100	125	150
Sensor	5	10	20	29	39	49	74	98	124	149

Según la tabla vemos que efectivamente los resultados son muy parecidos a los de su homólogo izquierdo, con un error de 1 o 2 cm. Al ser ambos sensores del mismo modelo y tener el mismo programa para cada uno de ellos es lógico pensar que los valores obtenidos tienen que proporcionar más o menos los mismos resultados.

7.3.1.2. Encoders

Los encoders son los encargados de llevar la cuenta del movimiento de los motores. Tenemos un total de cuatro parejas de encoders; dos por cada rueda situados arriba y debajo de la placa del robot. De este modo se puede tener una mayor precisión sobre la velocidad de la rueda en ese momento.

A medida que los motores giran, también lo hacen las ruedas a las que van sujetas. Estas ruedas tienen unos orificios equidistantes del centro de igual tamaño y

separados el mismo número de grados a lo largo de toda la periferia de manera que los encoders son capaces de captar el paso de luz a sombra que se produce entre ellas.

Al captar las diferencias de luz se realizan una serie de operaciones que permiten averiguar la velocidad a la que giran las ruedas y por tanto el movimiento del robot.

Para comprobar el correcto funcionamiento de los encoders dividiremos el estudio en función de las ruedas. De esta forma veremos en primer lugar el funcionamiento de los encoders de una de las ruedas y posteriormente haremos el mismo proceso en la otra.

Realizaremos varias pruebas para comprobar la precisión de los encoders:

- En primer lugar veremos si los encoders son capaces de contar correctamente el número de pasos de luz a sombra que se producen al dar una serie de vueltas a la rueda. Cada enconder lleva asociado un contador que aumenta su valor con cada paso de luz a sombra. Veremos si es capaz de contar el mismo número de incrementos a distintas velocidades del motor.
- Por otro lado veremos si es capaz de mostrar fielmente la velocidad a la que gira la rueda en función del número de contadores que hayan captado los encoders en un determinado tiempo.

7.3.1.2.1. Rueda izquierda

Lo primero que debemos comprobar es que los encoders captan perfectamente el paso de luz independientemente de la velocidad a la que gire la rueda.

Para realizar este estudio pondremos en marcha el motor y haremos girar a la rueda un total de diez vueltas y veremos el número de contadores que tiene cada uno. Este procedimiento lo realizaremos para un total de cuatro velocidades distintas equivalentes con el 25%, 50%, 75% y 100% de la potencia dada por los motores.

Colocaremos una pequeña marca en uno de los agujeros de la rueda que nos servirá como guía a la hora de contar el número de vueltas dada por la rueda *(Figura 124)*.

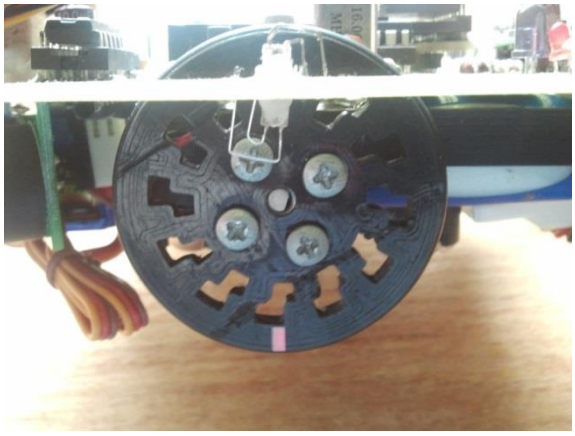


Figura 124: Prueba de contador de rueda

Las ruedas constan de un total de doce orificios tanto para el encoder superior como para el inferior. La diferencia entre ambos radica en que los del encoder inferior se encuentran desfasados un 50% con respecto al superior. De este modo cuando realice una vuelta completa ambos contadores habrán llegado a 24 (12 de luz y 12 de sombra). Tras diez vueltas el contador global tendrá que marcar un valor de 480.

Cada uno de los encoders cuenta con dos estados distintos: paso de luz a sombra y paso de sombra a luz. La captación de estas diferencias es lo que nos permite llevar la cuenta de lo que gira la rueda.

En total para esta prueba mostraremos los resultados correspondientes a cada uno de los estados que captan cada uno de los encoders. También mostraremos el valor del contador total al final de todas las vueltas.

1		25%	50%	75%	100%
Superior	Sombra-Luz	120	120	120	120
	Luz-Sombra	120	121	120	120
Inferior	Sombra-Luz	119	120	119	120
	Luz-Sombra	120	121	120	120
Total		479	482	479	480
2		25%	50%	75%	100%
Superior	Sombra-Luz	120	119	120	119
	Luz-Sombra	121	120	120	120
Inferior	Sombra-Luz	120	119	120	119
	Luz-Sombra	121	120	120	119
Total		482	478	480	477
3		25%	50%	75%	100%
Superior	Sombra-Luz	120	120	120	120
	Luz-Sombra	120	120	120	121
Inferior	Sombra-Luz	120	120	120	120
	Luz-Sombra	120	120	121	121
Total		480	480	481	482

Para mostrar los resultados, esta prueba la hemos realizado un total de tres veces para cada una de las velocidades para comprobar que los resultados no son fruto de la casualidad. En todos ellos los resultados han sido similares.

En ocasiones al detener el motor tras dar las diez vueltas (sobre todo cuando la velocidad de giro es mayor) los contadores no daban como resultado el valor de incrementos exacto. Debemos indicar que esto no es un error; lo que ocurre es que cuando gira el motor y detenemos manualmente su funcionamiento cuando ya ha realizado todas las vueltas, en ocasiones es difícil calcular el punto exacto al que debemos dejar la rueda para que el resultado sea el de los incrementos de diez vueltas exactas. Por lo tanto no es de extrañar que al detener el robot nos pasemos o nos quedemos algo cortos con respecto al número de incrementos que realmente

tendríamos que tener. Esto es lo que se puede ver reflejado en los resultados cuando no coincide con el número exacto que debería marcar el contador.

Una vez comprobado que los contadores de la rueda funcionan perfectamente, lo siguiente que debemos comprobar es que estos valores se utilizan correctamente para dar como resultado la velocidad de movimiento del robot y de giro de las ruedas.

Puesto que en nuestro caso no disponemos de un medidor que nos indique en tiempo real la velocidad a la que se mueven los motores, tendremos que comprobar los resultados obtenidos por pantalla con cálculos de pruebas reales.

En esta ocasión lo que haremos para ver si los datos son correctos será calcular el tiempo que tarda el robot en avanzar una cierta distancia. Como estamos calculando solamente una rueda, esta prueba la realizaremos viendo el número de veces que gira la rueda en un determinado tiempo. Sabiendo el perímetro de la circunferencia de la rueda, podemos averiguar la velocidad a la que se mueve.

Midiendo la circunferencia de la rueda comprobamos que ésta tiene una longitud de 13.2 cm/vuelta.

A continuación ponemos en marcha el motor y vemos cuantas vueltas ha girado la rueda en un espacio de 10 segundos. Con estos datos podemos calcular la velocidad y compararlos con los resultados obtenidos en el monitor.

$$v_{real} = 13.2 \frac{cm}{vuelta} * n \frac{vueltas}{10\ s}$$

Así mismo deberemos utilizar este valor y transformarlo en rpm para poder compararlo con el valor mostrado en la consola.

Partimos de una velocidad lineal y será necesaria pasarla a una velocidad de giro. Para ello será necesario utilizar la ecuación que relaciona ambas velocidades.

$$v = \omega * r$$

Donde *r* es el radio de nuestra rueda (en este caso 2.1 cm), *v* la velocidad lineal medida en cm/s y ω es la velocidad angular en rad/s.

Nuestro objetivo es hallar la velocidad angular en rpm (rev/min). Para ello deberemos pasar los rad/s a esta unidad. Una vuelta completa (o revolución) equivale a 2π rad. Por otro lado tenemos que 1 min son 60 s. De este modo tenemos:

$$\omega = [rpm] * 2\pi * \frac{1}{60}$$

Juntando ambas ecuaciones y despejando las rpm podemos calcular el valor de nuestra velocidad en la unidad adecuada.

$$[rpm] = \frac{60}{2\pi} * \frac{v}{2.1}$$

Realizamos las medidas y reflejamos todos los datos en la siguiente tabla.

	Vueltas/10 s	Vel. Real		Vel. Monitor [rpm]
		v [cm/s]	ω [rpm]	
25%	6.5	8.58	39.01	37.43
50%	9	11.88	54.02	53.26
75%	11.5	15.18	69.02	67.37
100%	14	18.48	84.03	79.89

Realizamos varias medidas para cada una de las velocidades para comprobar que los resultados son similares a los valores mostrados por Arduino. A medida que la velocidad de los motores aumenta, es más complicado calcular la velocidad real de forma exacta; aun así nos encontramos en un rango bastante cercano entre ambos.

7.3.1.2.2. Rueda derecha

El proceso que se realiza en la rueda derecha es el mismo que para el caso anterior. Tomaremos las medidas de los contadores a distintas velocidades y anotaremos los resultados obtenidos.

La tabla de los contadores nos queda en este caso:

1		25%	50%	75%	100%
Superior	Sombra-Luz	119	119	120	120
	Luz-Sombra	120	119	120	120
Inferior	Sombra-Luz	119	119	120	119
	Luz-Sombra	120	120	120	120
Total		478	477	480	479
2		25%	50%	75%	100%
Superior	Sombra-Luz	120	120	119	120
	Luz-Sombra	120	120	120	120
Inferior	Sombra-Luz	120	119	119	120
	Luz-Sombra	120	120	119	120
Total		480	479	477	480
3		25%	50%	75%	100%
Superior	Sombra-Luz	120	120	120	120
	Luz-Sombra	120	121	120	120
Inferior	Sombra-Luz	119	120	119	120
	Luz-Sombra	120	121	120	120
Total		479	482	479	480

Aquí podemos ver que al igual que antes, los resultados son similares y todos rondan los 480 pulsos correspondientes a diez vueltas completas de las ruedas. La variación de un par de pulsos por encima o debajo con respecto al objetivo corresponde a nuestra precisión a la hora de detener el motor.

A continuación realizamos las pruebas para comprobar las velocidades de giro de la rueda. Ambas ruedas tienen las mismas dimensiones y tomaremos el mismo tiempo de 10 segundos. Por tanto las operaciones serán igual que antes.

La tabla de velocidades de la rueda derecha nos queda de la siguiente forma:

	Vueltas/10 s	Vel. Real		Vel. Monitor [rpm]
		v [cm/s]	ω [rpm]	
25%	6.75	8.91	40.51	39.82
50%	9.5	12.54	57.02	54.29
75%	11	14.52	66.02	65.58
100%	13	17.16	78.03	75.69

Realizamos varias mediciones para todas las velocidades y comprobamos al igual que antes que los valores entre ambas formas de velocidad son similares entre sí, lo que indica que tenemos una buena aproximación. Si nos fijamos, las velocidades de los motores son distintas una de otra cuando le aplicamos el mismo valor de potencia. Esto ocurre puede ocurrir por varias razones entre las que podría ser que uno recibiera más energía que el otro o bien que aun tratándose del mismo modelo de motor podemos encontrarnos con variaciones técnicas.

7.3.2. Funciones avanzadas

Una vez comprobadas las funciones básicas de las que consta nuestro robot, veremos aquellas funciones más complejas y que unifican en un mismo programa las anteriores.

Al igual que en el caso anterior veremos si funcionan como se espera de ellas. En esta parte se encuentran los movimientos de giro y avance del robot.

7.3.2.1. Giro del robot

En primer lugar comprobaremos la función correspondiente al giro del robot. Ya se habló en su momento del funcionamiento de esta función. En esta ocasión comprobaremos si los giros que realiza el robot son los apropiados.

Veremos el giro hacia cada uno de los lados (izquierda y derecha) y comprobaremos que es capaz de girar un número de grados determinado.

7.3.2.1.1. Giro a izquierdas

En primer lugar comprobaremos el giro del robot hacia el lado izquierdo. El proceso que sigamos será el mismo para la otra dirección.

Para ver el correcto funcionamiento de esta función la programaremos para que gire hacia el lado izquierdo y comprobaremos el movimiento que realiza para ángulos de diferente amplitud hasta un máximo de una vuelta completa.

Es importante tener una referencia con la que comparar el ángulo realizado. Para saber su precisión colocaremos al robot sobre una superficie en la que se aparezcan reflejados los distintos ángulos de una circunferencia.

En la siguiente imagen podemos ver en qué consiste la prueba *(Figura 125)*.

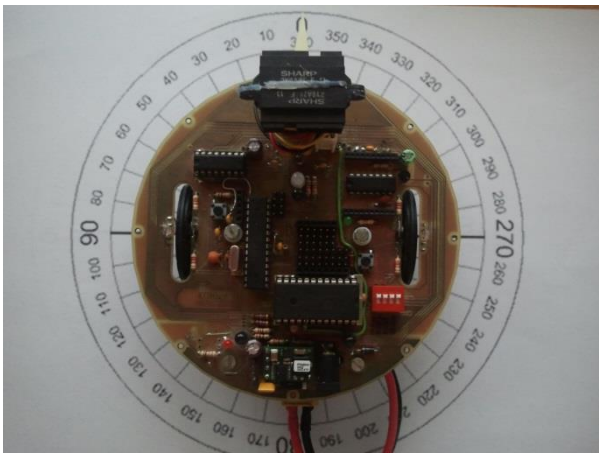


Figura 125: Prueba de giro del robot

Los resultados obtenidos están recogidos en una tabla en la que se indica el giro realizado por el programa y la posición real que ha adoptado el robot para ese ángulo.

Ángulo solicitado	Giro del robot		
10°	8°	10°	11°
20°	18°	20°	21°
30°	29°	30°	33°
40°	38°	40°	41°
45°	42°	45°	48°
60°	58°	60°	62°
70°	68°	70°	73°
90°	87°	90°	93°
135°	133°	135°	137°
180°	176°	180°	181°
225°	221°	225°	226°
270°	267°	270°	272°
315°	312°	315°	317°
360°	358°	360°	363°

En este caso hemos decidido poner a prueba los ángulos más comunes que podemos realizar con el robot. Esto lo hemos realizado varias veces para los mismos ángulos para ver lo que pueden variar entre sí.

Tenemos que tener en cuenta el error que se produce cuando los ángulos introducidos no son múltiplos de 3.5° (este es el ángulo de giro mínimo para este diseño concreto de ruedas); como ya indicamos en el apartado de programación el error producido varía entre 0.5° y 1.5°. Si encadenásemos varios ángulos seguidos de la misma amplitud el error se iría acumulando.

Como se puede observar en la tabla los ángulos obtenidos oscilan como esperábamos alrededor del valor pedido. Estos resultados están también influidos por la posición de las ruedas; ya que en función de cuando empieza a contar el programa puede que realice un pequeño incremento más o menos. Debemos indicar también que en ocasiones puntuales el ángulo de giro se ha sobrepasado o se ha quedado corto algo más de lo esperado.

Tenemos que tener en cuenta que el giro del robot funciona a base de incrementos del contador. Como indicamos anteriormente en el apartado de programación cada incremento de giro de la rueda corresponde con un ángulo base de 3.5°. Esto significa que para aquellos ángulos que no sean múltiplos de este valor, obtendremos un giro aproximado al ángulo pedido.

Esto se puede ver reflejado en los datos mostrados. Si el ángulo de incremento fuese menor, la precisión sería mayor. Para ello habría que tener unas ruedas con un mayor número de orificios.

Un detalle a tener en cuenta es que si la velocidad de giro es muy rápida, se pueden obtener resultados con una precisión algo menor que con velocidades más bajas. Esto ocurre por la inercia de los motores cada vez que se conectan y se desconectan para realizar un cierto giro.

También es importante señalar que ambos motores deben estar calibrados a una velocidad similar, ya que de lo contrario el robot no giraría sobre sí mismo; si no que realizaría pequeños desplazamientos circulares.

7.3.2.1.2. Giro a derechas

De la misma forma que para el caso anterior, veamos ahora cómo se comporta el programa para el giro en el otro sentido.

Debido a que el programa es el mismo esperamos obtener resultados similares. A continuación mostramos la tabla con los resultados de los giros obtenidos para este caso.

Ángulo solicitado	Giro del robot		
10°	7°	10°	11°
20°	19°	20°	21°
30°	29°	30°	32°
40°	37°	40°	41°
45°	43°	45°	48°
60°	59°	60°	63°
70°	68°	70°	72°
90°	88°	90°	93°
135°	134°	135°	138°
180°	177°	180°	181°
225°	222°	225°	227°
270°	267°	270°	273°
315°	314°	315°	317°
360°	357°	360°	362°

Efectivamente podemos ver que al igual que antes los valores se encuentran en un rango cercano al ángulo solicitado. Esto conlleva a que también sufre de las mismas limitaciones que su semejante a izquierdas.

7.3.2.2. Avance a punto concreto

Esta prueba consiste en probar la precisión del robot en tramos rectos. Como indicamos anteriormente existen dos modos de movimiento rectilíneo del robot: uno consistente en un avance continuo y otro de avance a una distancia determinada. Es de este último caso del que vamos a hablar a continuación.

Para la realización de esta prueba hemos utilizado el metro que se usó en las medidas de los sensores. En esta ocasión colocaremos al robot en una posición inicial y le programaremos distintas distancias para que las recorra. Cuando el robot alcance su objetivo se detendrá. En ese momento tomaremos la medida de la posición final y la compararemos con la distancia solicitada en el programa.

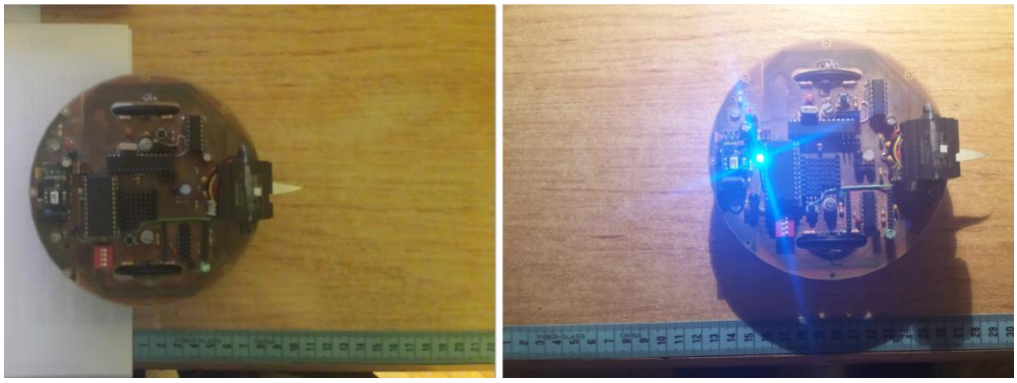


Figura 126: Prueba de avance a punto concreto

En la imagen anterior se aprecia la prueba del desplazamiento del robot para una distancia de 20 cm *(Figura 126)*.

Regulando la velocidad de los motores para que ambos avancen a una distancia similar comprobamos que la precisión es bastante buena. El robot se acerca mucho a su objetivo sin apenas error. Sin embargo a medida que la distancia solicitada es mayor, esta precisión se va perdiendo debido precisamente a esa diferencia de velocidad en los motores. Estos hacen que el robot se vaya desviando poco a poco, ya que al girar más uno que otro, el robot deja de avanzar en línea recta para trazar una cierta curvatura en su recorrido hacia el lado que se mueve más despacio.

Es precisamente este el mayor problema de esta función: la linealidad en el recorrido del robot. Cuanto mayor sea la diferencia entre velocidades de los motores, mayor será el error y el desvío del robot.

A continuación mostramos una tabla con alguno de los resultados obtenidos para distintas distancias.

Distancia solicitado	Distancia real
10 cm	10 cm
20 cm	20 cm
30 cm	29.8 cm
40 cm	40 cm
50 cm	49.6 cm
100 cm	99.3 cm
150 cm	148.7 cm

Estas medidas han sido tomadas ajustando los motores con una velocidad aproximada del 25%. Como se puede comprobar la precisión es bastante buena; acercándose mucho a la distancia solicitada. Como hemos comentado antes, a medida que la distancia es mayor el error también aumenta, pero se puede minimizar si conseguimos un buen equilibrio de velocidad entre ambos motores.

Veamos qué ocurriría ahora con una velocidad mayor.

Distancia solicitado	Distancia real
10 cm	11.3 cm
20 cm	20.8 cm
30 cm	30.5 cm
40 cm	40.5 cm
50 cm	50.1 cm
100 cm	99.5 cm
150 cm	149.1 cm

Estas medidas han sido tomadas con una velocidad aproximada del 75%. En esta ocasión podemos comprobar como en las distancia más cortas el error es mayor que en el caso anterior y que este error va disminuyendo a medida que las distancias son mayores.

Esto ocurre debido a que la velocidad de los motores es mayor, lo que se traduce en una reacción más tardía para detener los motores. Debido a la inercia que llevan hasta detenerse, avanzan más de lo esperado. Por eso a medida que la distancia aumenta el error se va corrigiendo haciendo que en medidas mayores la precisión sea algo mayor que con velocidades más pequeñas.

7.3.2.3. Mapeado

En este último apartado vamos a centrarnos en mostrar el funcionamiento de las funciones cuyos datos permiten generar los distintos mapas en 2D en los cuales se va a mover nuestro robot.

Es importante señalar que al realizar las funciones los datos que obtenemos corresponden a una serie de puntos que deben situarse en un eje de coordenadas X e Y. Para ello recogeremos todos estos puntos y los representaremos en una gráfica. La unión de todos esos puntos será lo que genere un mapa básico del entorno. Usaremos las que nos proporciona Microsoft Excel para este fin.

Como hemos dicho en otras ocasiones tenemos dos tipos de funciones diferentes para representar los mapas: una con el robot girando sobre sí mismo y otra con el robot en movimiento.

7.3.2.3.1. Mapeado estático

El primer caso que vamos a ver consiste en dejar al robot en un entorno y esperar a que realice un escaneo de 360° de toda la estancia.

Estableceremos un ángulo que determine la amplitud entre escaneos. Cuanto menor sea este ángulo, más medidas tomará el robot hasta completar una vuelta y, por tanto más preciso será el mapa que genere. Los distintos puntos estarán más próximos entre sí y menor será el error que se produzca por dejar elementos sin escanear.

Debemos indicar también que la precisión será menor cuanto más alejados se encuentren los objetos del robot. Los sensores de ultrasonidos tienen un alcance aproximado de 150 cm; el sensor infrarrojo tiene un alcance menor con unos 80 cm más o menos.

Para ver el funcionamiento de esta función situaremos al robot en un entorno real; utilizaremos un espacio cerrado cuyas paredes se encuentran a cierta distancia del

robot. El objetivo es que el robot sea capaz de captar la posición de estas paredes y lo refleje con los sensores.

Figura 127: Ejemplo de entorno de mapa

En la imagen anterior podemos ver un ejemplo cualquiera en el cual se puede colocar al robot *(Figura 127)*. Será aquí donde realicemos las pruebas de mapeado para comprobar si somos capaces de generar un mapa.

Como tenemos tres sensores y el robot únicamente gira sobre su eje, tendremos que generar un total de tres mapas distintos: uno por sensor.

Introduciendo la función en el programa y conectando el robot obtenemos los distintos puntos. Estos quedan reflejados en la siguiente tabla.

Sensor IR		Sensor USD		Sensor USI	
X	Y	X	Y	X	Y
0	85	15	25	-27	46
-18	56	17	79	-33	29
-43	59	-8	80	-28	13
-35	25	-33	73	-42	4
-41	13	-37	41	-29	-6
-46	0	-33	19	-26	-15
-33	-11	-41	9	-29	-33
-37	-27	-39	-4	-12	-26
-25	-34	-38	-17	-3	-30
-14	-44	-31	-28	10	-45
0	-51	-22	-38	16	-27
17	-51	-11	-53	25	-23
42	-57	4	-36	41	-18
49	-36	32	-72	72	-8
48	-15	54	-59	68	15
60	0	75	-44	25	15
49	16	39	-8	33	36
35	25	30	3	12	26
14	19	27	12	4	40
12	36	21	19	-12	58

En la tabla anterior vemos las posiciones para un total de 20 datos distintos; esto supone una amplitud de 18° (como los giros se realizan en múltiplos de 3.5° la aproximación de giro será de unos 17.5°). Cuantos más puntos tengamos más fiable será el mapa.

Fijándonos en la forma básica de cómo es el recinto cerrado en el cual se ha colocado al robot podremos compararlo con los mapas creados.

Una vez tengamos los distintos puntos es el momento de trasladarlos a un eje de coordenadas. Como se indicó anteriormente usaremos las gráficas de Excel para representarlos. El resultado nos queda:

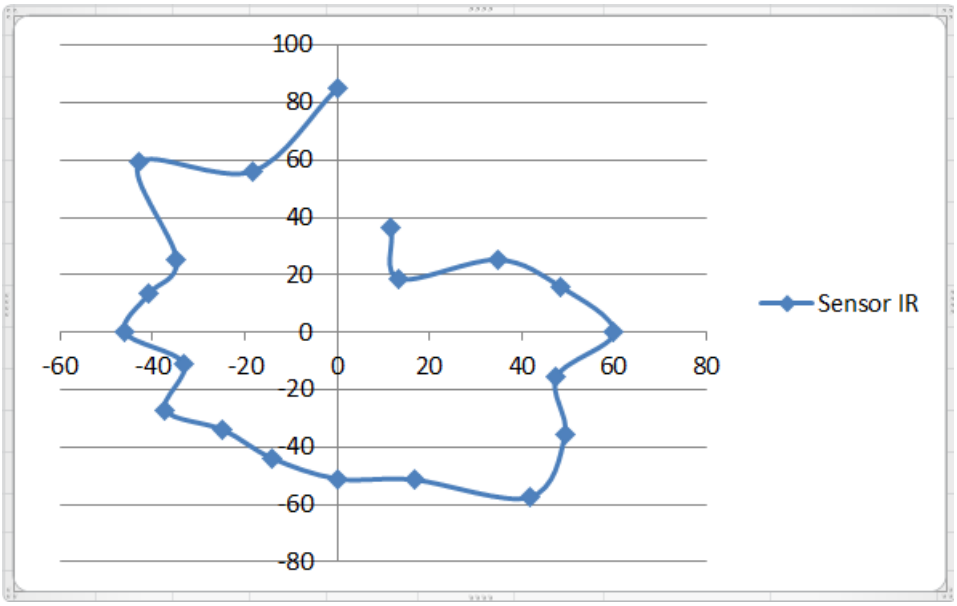


Figura 128: Mapa sensor infrarrojos. Caso 1

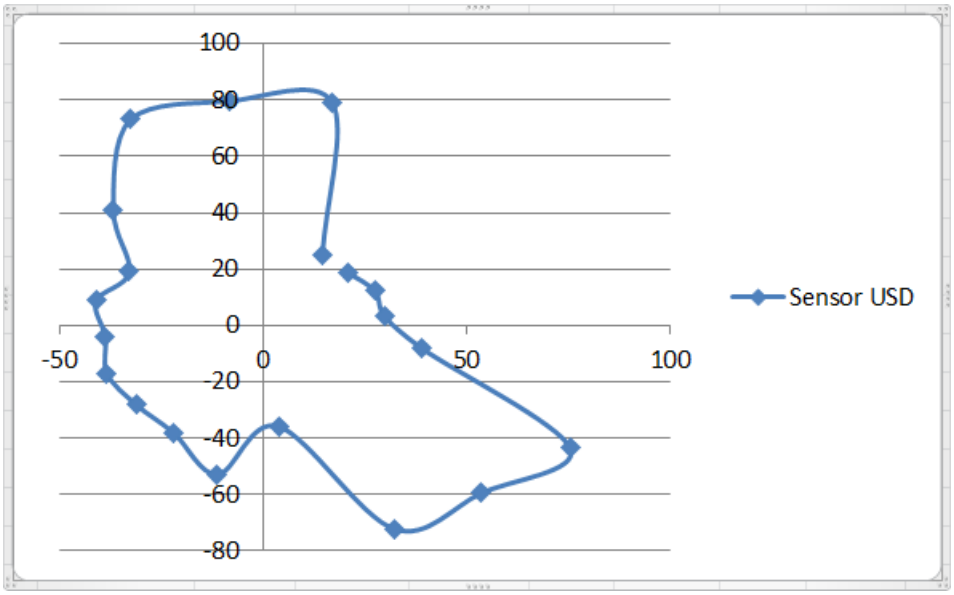


Figura 129: Mapa sensor ultrasonidos derecho. Caso 1

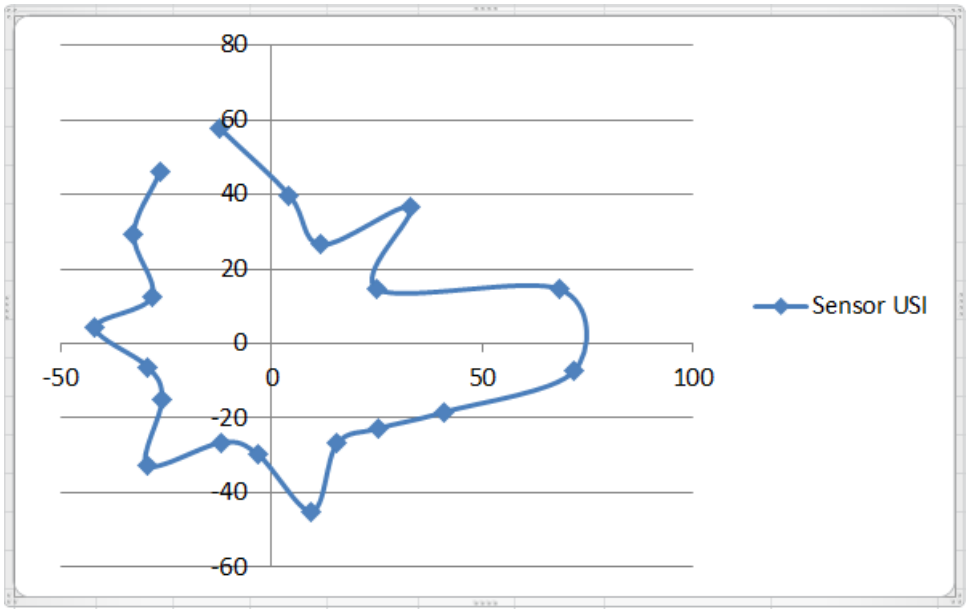


Figura 130: Mapa sensor ultrasonidos izquierdo. Caso 1

En estos mapas debemos situar al robot en el eje [0,0]. Si unimos el primer punto con el último nos queda un área cerrada. Esta área es el que delimita los obstáculos y paredes con los que se ha encontrado el robot.

Podemos cambiar del número de datos solicitados al programa para así tener más puntos y hacer que el mapa sea más preciso; pero de todas formas seguiría siendo un mapa básico que puede servir de referencia para tener una idea de cómo es el entorno en el cual se encuentra nuestro robot.

Se puede comprobar en los tres casos que la forma básica obtenida por los sensores se corresponde con el espacio real en el que se ha situado el robot.

Otra cosa a tener en cuenta es que en ocasiones podemos recibir una medida errónea; bien obteniendo un valor más cercano o más lejano al real. Esto también influye a la hora de crear el mapa ya que se producen discrepancias con la realidad. Lo podemos ver a simple vista en los mapas creados; se aprecia a simple vista como existen algunos puntos que no se corresponden.

El sensor más preciso de los tres ha resultado ser el sensor de infrarrojos *(Figura 128)*; se puede comprobar como los valores obtenidos son menos erráticos que los obtenidos con los de ultrasonidos derecho *(Figura 129)* o izquierdo *(Figura 130)*. La distancia máxima que capta el sensor de infrarrojos es menor pero más precisa; únicamente se vería afectada por efectos de la luz y clima tales como luz directa o polvo.

En el caso de los sensores de ultrasonidos estos errores pueden deberse a que la superficie sobre la cual rebota el pulso con el que mide no se encuentra perpendicular al sensor, con lo que el eco no se recibe correctamente si tenemos un cierto ángulo de

reflexión. Esto nos lleva a pensar que los sensores de ultrasonidos no son los más apropiados para realizar mapeados.

7.3.2.3.2. Mapeado en movimiento

El otro caso de mapeado es aquel que crea el robot cuando se desplaza. A medida que se mueve va marcando el camino que sigue y los objetos que detecta.

En esta ocasión el robot comienza situado de forma paralela a la pared y lo que realiza en esta función es un avance continuo siguiendo esa pared. Para ello nos basamos en el sensor de infrarrojos el cual utilizamos como referencia.

Gracias a que podemos dirigir el sentido de hacia dónde apuntar gracias al servo, colocamos al sensor mirando de forma continua a la pared. De este modo medimos la distancia a la cual se encuentra en el momento de comenzar a moverse y procuramos que el robot continúe a esa distancia a medida que se desplaza.

De forma continua tomamos medidas con los sensores para conocer en todo momento la situación del robot y saber así como debe moverse.

Para crear los mapas nos hemos basado en la posición en la que se encuentra el robot en cada momento. A diferencia del caso anterior en el cual el robot no se movía del sitio ya que solo giraba sobre sí mismo, aquí se desplaza en función de lo que detecten los sensores.

Tenemos en esta ocasión un total de cuatro mapas; aparte de los tres mapas pertenecientes a cada uno de los tres sensores, aquí nos encontramos con otro mapa que representa la posición del robot en los puntos clave en los cuales gira.

Es a partir de este último mapa que nos basamos para crear los otros debido a que lo usan como referencia. Utilizaremos el mismo entorno que en el caso 1 para recrear los mapas.

Para una mayor comprensión de los mapas de los sensores iremos viendo cada uno junto con el mapa de posición del que proceden.

Veamos en primer lugar los resultados del sensor de infrarrojos.

Pos	X	0	0	-40	-40	-102	-102	-9						
	Y	0	110	110	150	150	4	4						
IR	X	16	16	17	0	-40	-23	-23	-23	-74	-102	-115	-119	-9
	Y	23	57	110	127	171	110	144	150	162	168	116	4	-14

Cómo podemos comprobar es que los puntos pertenecientes a la posición del robot son menos numerosos que los proporcionados por el sensor IR. Esto ocurre porque como ya hemos explicado anteriormente, el sensor toma valores intermedios con una

determinada frecuencia de tiempo (3 segundos en este caso) de tal manera que se van incorporando al mapa en los tramos rectos.

A continuación mostramos como nos quedarían representadas *(Figura 131)*.

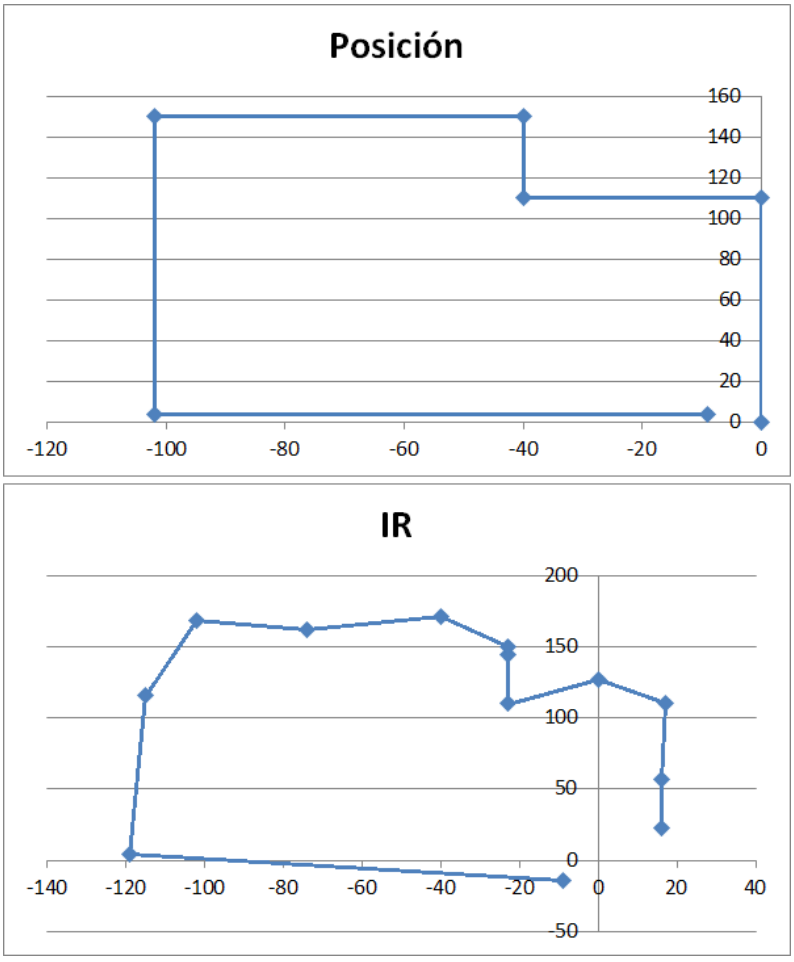


Figura 131: Mapa sensor infrarrojos. Caso 2

En este caso los puntos que recoge el robot no van correlativos; en función de la pared que esté señalando, el punto representado estará en un sitio u otro.

Debido a los giros a izquierda y derecha, en ocasiones el sensor muestra el valor de la posición de otras paredes. Ordenándolos y uniéndolos mediante líneas podemos hacernos una idea del contorno de la habitación que ha recorrido.

Como podemos comprobar en el primer mapa, el robot ha ido avanzando por la estancia y girando en los momentos clave. Como hemos definido que el robot realice giros de 90°, tenemos una buena aproximación al conjunto de la habitación.

Esto es una ventaja y un inconveniente. Cuando nos encontramos ante habitaciones de este estilo, las esquinas suelen ser ángulos rectos. En estos casos la fiabilidad es mayor.

En el caso de que el espacio por el que se mueve tuviese otro tipo de ángulos (agudos u obtusos), la precisión seria mucho menor y se reflejaría en la creación de los mapas, ya que para resolver este tipo de paredes el robot realizaría movimientos escalonados con giros continuos que lo alejarían y acercarían continuamente a la pared.

Otro problema que podemos encontrarnos es en el propio movimiento del robot. Cabe la posibilidad de que aunque el robot se mueva y gire de forma recta, es posible que las ruedas no hagan lo mismo. Puede darse el caso que alguna de las ruedas se escurran según se mueven. Esto hace que los resultados obtenidos sean erróneos puesto que el robot piensa que está en una posición cuando en realidad está en otra diferente.

Sin embargo cuando el robot no tiene ninguno de estos problemas, llega a seguir bastante bien la periferia de la habitación.

Podemos comprobar como en el mapa perteneciente al sensor IR tenemos un contorno parecido al mapa creado por la posición del robot. Debido a que este sensor se encuentra continuamente mirando hacia la pared de forma paralela al movimiento del robot, detecta estas distancias bastante bien.

Como resultado obtenemos un mapa en el que se incluye a la posición del robot la separación hasta la pared. Los puntos del mapa son el resultado de la posición del robot en ese instante más el valor captado por el sensor.

A continuación mostraremos los resultados obtenidos por el sensor de ultrasonidos derecho. Al igual que antes mostramos en primer lugar el valor de los resultados obtenidos.

Pos	X	0	0	-39	-39	-97	-97	5						
	Y	0	108	108	151	151	-11	-11						
USD	X	13	16	0	-23	-67	-82	-26	-26	-39	-103	-97	-102	5
	Y	46	87	108	121	126	133	131	166	151	168	151	-20	-11

Aquí nos encontramos con el mapa creado por el sensor de ultrasonidos derecho *(Figura 132)*. Dicho sensor se encuentra en el lado de la pared, por lo que al igual que en el caso anterior las distancias que detecta son relativamente cortas por su cercanía a la pared.

Sin embargo aquí podemos comprobar que el resultado difiere un poco en algunos puntos. Se puede comprobar cómo en alguna ocasión a recibimos un valor erróneo y eso se refleja en el mapa final.

La precisión es menor que en el caso anterior debido al tipo de sensor y a la localización del mismo, aunque nos da una idea del entorno. Existen zonas con puntos más juntos y otros con puntos más dispersos, esto es debido a la posición en la que se encontraba en cada uno de los instantes en los que guardaba un dato.

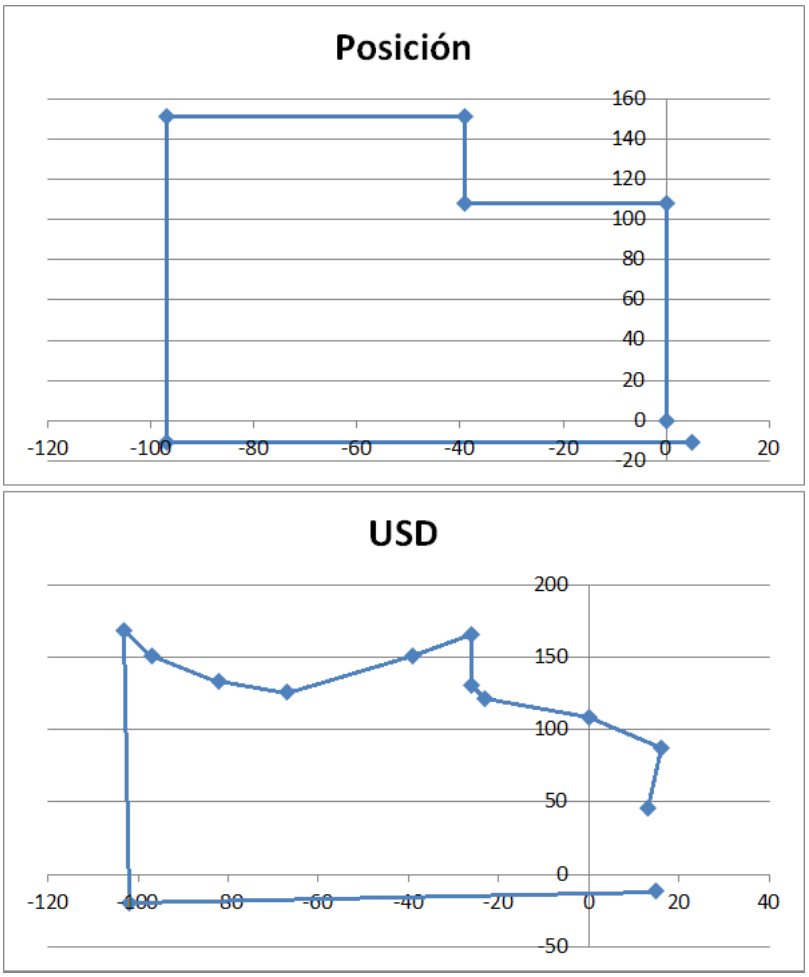


Figura 132: Mapa sensor ultrasonidos derecho. Caso 2

Otra cosa en lo que nos podemos fijar es que en ocasiones hay puntos del mapa de posición y del sensor USD que tienen los mismos valores. Esto ocurre porque a la hora de girar el robot, éste lo ha hecho muy próximo a la pared y el valor captado en ese momento por el sensor es 0, por lo que no añade nada más al mapa.

Finalmente veamos los resultados mostrados por el último sensor correspondiente con el sensor de ultrasonidos izquierdo.

Pos	X	0	0	-41	-41	-105	-105	12							
	Y	0	110	110	149	149	3	3							
USI	X	-14	-25	-5	-24	-59	-66	-55	-55	-46	-96	-75	-112	-101	18
	Y	48	101	119	97	96	96	134	169	158	118	149	145	-4	7

Del mismo modo que en los otros dos casos anteriores, pasemos a mostrar los mapas generados por estos puntos.

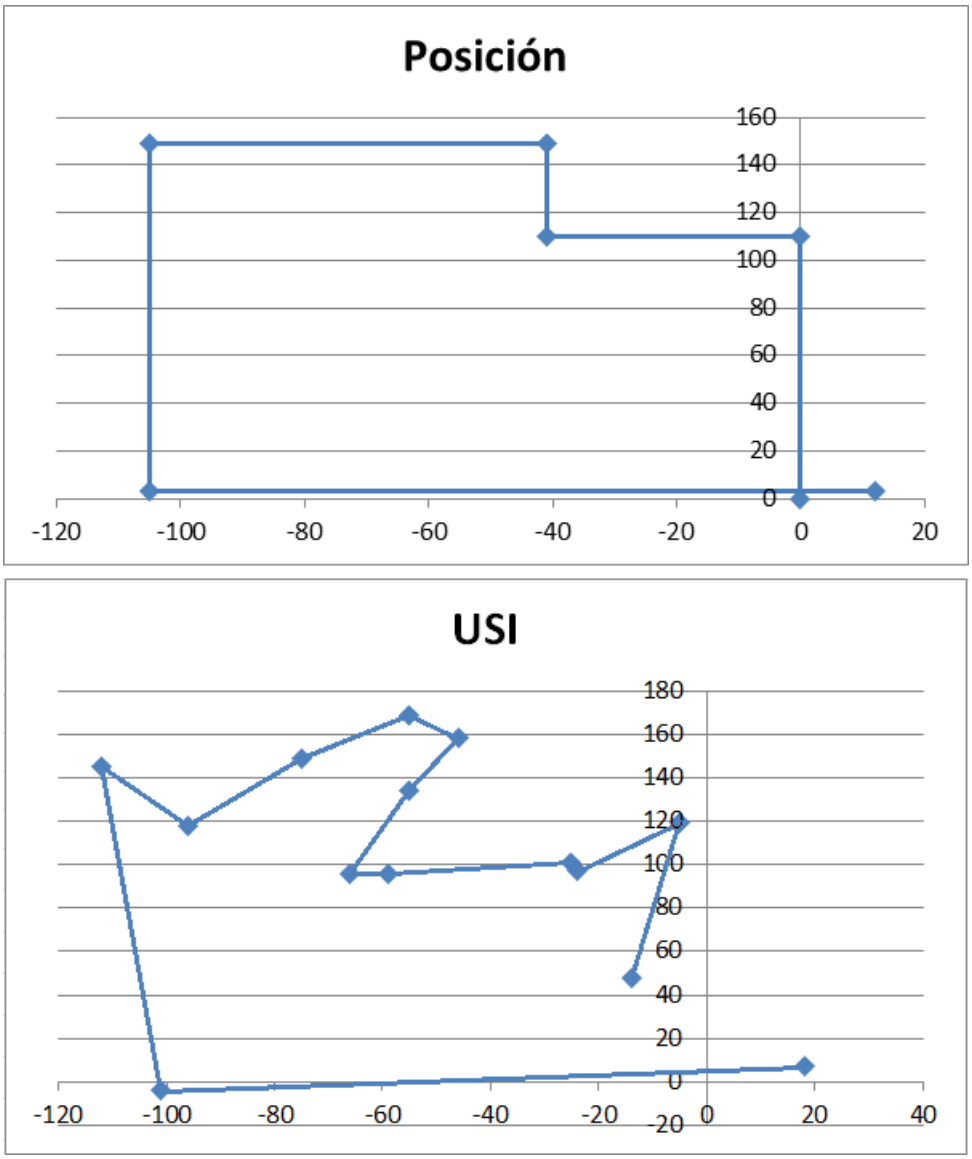


Figura 133: Mapa sensor ultrasonidos izquierdo. Caso 2

El mapa generado por este sensor es el peor de los tres *(Figura 133)*. Es el que menos se refleja con la realidad.

Si el obstáculo está muy lejos, el sensor no lo detecta y nos da valores erróneos. Esto se puede comprobar con los puntos intermedios de la gráfica. Los puntos más fiables son aquellos que se encuentran en las distintas esquinas, cuando el robot debe girar ya que detecta mejor la pared.

En nuestro caso como estos valores falsos los consideraba a una distancia aproximada de 60 cm, el robot generaba estos puntos como si fuese una pared que se encuentra en el mismo sentido que la dirección del robot dando esos puntos consecutivos. Al tratarse del sensor USI los valores que obteníamos los restaba a las puntos referidos al mapa de la posición del robot, creando así paredes más pequeñas.

Lo deseable hubiese sido que detectase las paredes más lejanas a medida que se mueve, puesto que son aquellas con las que tendría que toparse el sensor. Estos puntos en orden habrían estado más dispersos, pero mirando el conjunto final daría como resultado un mapa parecido al original.

Aunque nuestro mapa tiene una forma similar al real no es muy válido, sobre todo en los puntos intermedios entre giros por no detectar distancias tan lejanas.

La distancia máxima de detección es de 150 cm, pero para ello el sensor debe estar de frente al objeto y no moverse. En nuestro caso el eco se pierde o se desvía (debido al ángulo de reflexión de las ondas) rebotando en otros sitios dando lugar a diversos errores como ya hemos comentado anteriormente al hablar de las ventajas e inconvenientes de estos sensores.

Podemos concluir que este tipo de sensores no es el más apropiado para crear mapas.

8. Presupuesto

8.1. Proveedores

Este proyecto consta de un gran número de componentes de diversa índole. Es importante distinguir entre elementos software y hardware.

Para la construcción del robot se han usado una serie de programas informáticos que nos han permitido diseñar y programar diversas partes. El software no ha requerido de ningún tipo de inversión económica puesto que se han utilizado programas de software libre.

En la parte de hardware la mayoría de componentes son de tipo electrónico, aunque también nos encontramos con elementos como la placa que hemos encargado fabricar y el plástico necesario para imprimir el chasis, entre otros.

Las tiendas que nos han aportado los componentes más importantes han sido aquellas que tenían servicio web. Esto nos permite aumentar el rango de sitios distintos en los que buscar los diferentes precios y comparar entre ellos. En ocasiones hemos tenido que comprar componentes a tiendas distintas a las europeas, con el consiguiente cambio de moneda.

Los componentes más comunes (tales como resistencias, condensadores…) pueden adquirirse en cualquier tienda de componentes electrónicos, aunque también pueden adquirirse de forma on-line en el caso de no tener ninguna cerca.

Las tiendas en las que hemos encargado componentes han sido:

- RS
- Farnell
- HobbyKing
- Dealextreme
- iStore
- Pololu

8.2. Presupuesto

En este apartado estableceremos el presupuesto total del robot una vez construido. Tendremos en cuenta los componentes de cada una de las partes de los que consta y desarrollaremos el presupuesto en distintos apartados para una mejor visión global.

Indicaremos donde se ha obtenido cada uno de los componentes, así como su precio en el momento de la compra. En el caso de que construya otro modelo igual o similar al del presente proyecto, el valor de las piezas podría variar en función de las ofertas de las distintas tiendas dependiendo de cuando se adquieran.

Se han comprado varias piezas extra en algunos casos que finalmente no han sido colocadas en el robot; bien por servir como piezas de repuesto en el caso de que alguna no funcionase; bien por venderse en distintos packs de una misma pieza (en este caso solo consideraremos una de las piezas), o bien como piezas que podrían incluirse en una futura ampliación del proyecto. Estas piezas extra no se encuentran incluidas en el presupuesto.

Las tiendas en las que hemos adquirido nuestras piezas son orientativas; ya que no son únicas y siempre pueden encontrarse diferentes tiendas con distintos precios para los mismos componentes, por lo que ya es decisión personal de cada uno elegir dónde comprarlos.

En primer lugar incluimos en este apartado el software utilizado para desarrollar cada parte y la página web en donde podemos obtenerlos.

Software					
Nº	Nombre/Descripción	Tienda/Proveedor	Precio	Cant.	Total
1	KiKad	http://www.kicad-pcb.org/display/KICAD/KiCad+EDA+Software+Suite	0	1	0
2	Qt Creator	http://qt.digia.com/	0	1	0
3	OOML2	http://iearobotics.com/oomlwiki/doku.php?id=start	0	1	0
4	OpenScad	http://www.openscad.org/	0	1	0
5	Python	http://www.python.org/download/	0	1	0
6	Replicatorg	http://replicat.org/	0	1	0
7	Pronterface	https://github.com/kliment/Printrun/archive/master.zip	0	1	0
8	Slic3r	http://slic3r.org/	0	1	0
9	Arduino	http://arduino.cc/en/Main/Software	0	1	0
	Total		0		

A continuación mostramos todas aquellas piezas incluidas en el robot y que han sido adquiridas mediante tiendas on-line.

Hardware					
Nº	Nombre/Descripción	Tienda/Proveedor	Precio €	Cant.	Total €
1	Cable USB mini y conector Foca	http://imall.iteadstudio.com/im120525005.html	6,3	1	6,3
2	Microcontrolador ATMEGA168P AVR 32K FLASH 2K SRAM	http://es.rs-online.com/web/p/microcontroladores/6962260/	3,99	1	3,99
3	Multiplex/Demultiplex Analog CD4067BE-LOGIC, 16-CH	http://es.farnell.com/texas-instruments/cd4067be/ic-switch-mux-demux-bus/dp/1470913	1,49	1	1,49
4	Equilibrador y cargador Turnigy 2S-3S	http://www.hobbyking.com/hobbyking/store/__7637__Turnigy_balancer_Charger_2S_3S.html	3,4	1	3,4
5	Batería LiPo Pack Turnigy 1300mAh 2S 20C	http://www.hobbyking.com/hobbyking/store/uh_viewItem.asp?idProduct=10368	5,4	1	5,4
6	Convertidor DC-DC 2.25A PTH08000WAZT	http://es.rs-online.com/web/p/convertidores-dc-dc/6201136/	8,71	1	8,71
7	Detector de tensión 2.7V TC54VN2702EZB	http://es.rs-online.com/web/p/supervision-de-tension/2070051/	0,5	1	0,5
8	Fusible PTC reajustable 1.1 A radial RKEF110	http://es.rs-online.com/web/p/fusibles-con-pines-de-conexion-rearmables/7127499/	0,31	1	0,31
9	Interruptor de 4 vías, dil, slice 78B04T	http://uk.farnell.com/grayhill/78b04t/switch-dil-slide-4way/dp/9479040?Ntt=947-9040	1,14	1	1,14
10	Módulo XBee, antena de alambre, XB24-AWI-001, 1 MW	http://es.farnell.com/digi-international/xb24-awi-001/wire-antenna-module-xbee-1mw/dp/1337912?Ntt=1337912	21,44	2	42,88
11	Buffer 3 estados	http://es.rs-	0,47	2	0,94

	cuádruple 74ABt125	online.com/web/p/drivers-de-linea-bufer/1775913/			
12	Módulo ultrasónico sin zona ciega SDM-IO	http://store.iteadstudio.com/index.php?main_page=product_info&products_id=415	8,32	2	16,64
13	Sensor de distancia analógico GP2Y0A21YK0F (de 10cm a 80cm)	http://es.rs-online.com/web/p/sensores-fotoelectricos/6666564/	9,12	1	9,12
14	Fototransistor PT480F	http://es.rs-online.com/web/p/fototransistores/315-860/	0,36	4	1,44
15	Led IR de Vista lateral de emisión, GL4800E0000F	http://es.rs-online.com/web/p/ir-led/6666476/	0,7	4	2,8
16	Micro motor-reductor de metal 250:1	http://www.pololu.com/catalog/product/995	11,96	2	23,92
17	Emisor infrarrojo, 950 nm, 5mm LD271	http://es.rs-online.com/web/p/ir-led/6548362/	0,25	3	0,75
18	Driver de motor de cuatro canales L293DNE	http://es.rs-online.com/web/p/control-de-movimiento-de-motores/0526868/	2,69	1	2,69
19	Mini Servo TowerPro SG90 9G con accesorios	http://dx.com/es/p/towerpro-sg90-9g-mini-servo-with-accessories-12859	2,64	1	2,64
20	Switch SPTD-CO	http://es.farnell.com/eao/09-03290-01/slide-switch-spdt-vert/dp/674345	1,82	1	1,82
21	Par de ejes de montaje universales de aluminio Pololu	http://www.pololu.com/catalog/product/1078	4,28	1	4,28
	Total		141,16		

Mención aparte tiene el hilo de plástico usado para la construcción del chasis. Si bien el hilo se vende en bobinas de gran tamaño, nosotros no hemos usado todo el rollo para fabricarlo. Consideraremos por tanto la parte proporcional usada.

Nº	Nombre/Descripción	Tienda/Proveedor	Precio €	Cant.	Total €
1	Bobina de plástico negro ABS	http://shop.reclone3d.com/es/abs-3mm-bobinas-de-2-kg/84-bobina-de-abs-3mm-23-kg-rojo.html	34,95	1	34,95

Este es el precio total de una bobina de 2.3 kg. Como hemos indicado anteriormente el total de las piezas (ruedas y chasis) tienen un peso total aproximado de 25 g.

Tendremos en cuenta por lo tanto solo el precio del plástico gastado para un robot. De este modo tenemos que el precio del plástico usado ha sido:

Tamaño (g)	Precio €
2300	34,95
25	0,38

Los siguientes componentes son más genéricos y pueden adquirirse fácilmente en cualquier tienda de electrónica.

Otros elementos electrónicos					
Nº	Componente	Valor/Descripción	Precio	Cant.	Total
1	Condensador	100 nF	0,13	7	0,91
2		22 pF	0,06	2	0,12
3		8,2 pF	0,07	1	0,07
4		100 uF	0,13	2	0,26
5		10 uF	0,12	2	0,24
6		1 uF	0,08	1	0,08
7	Resistencia	15 KΩ	0,04	1	0,04
8		10 KΩ	0,04	3	0,12
9		2,2 KΩ	0,04	4	0,16
10		1,5 KΩ	0,04	2	0,08
11		1 KΩ	0,04	1	0,04
12		500 Ω	0,03	1	0,03
13		348 Ω	0,03	1	0,03
14		330 Ω	0,03	1	0,03
15		220 Ω	0,03	1	0,03
16		180 Ω	0,03	2	0,06
17		120 Ω	0,03	1	0,03
18		20 Ω	0,03	1	0,03
19	Bobina	10 uH	0.45	1	0.45
20	Oscilador	16 MHz	0.1	1	0.1
21	Led RGB		0.1	1	0.1
22	Interruptor	Push Button	0.15	1	0.15
23		Switch	0.45	1	0.45
	Canica	Metálica	0.7	2	1.4
	Goma rueda		0.5	2	1
	Total		6.55		

A continuación mostramos aquí todos los elementos que finalmente no forman parte del proyecto pero que fueron considerados en algún momento del desarrollo. Estos componentes no forman parte del presupuesto final.

Hardware descartado					
Nº	Nombre/Descripción	Tienda/Proveedor	Precio €	Cant.	Total €
1	Regulador lineal MCP1700-3302E/TO 3.3V	http://uk.farnell.com/microchip/mcp1700-3002e-to/ic-v-reg-ldo-250ma-smd-to-92-3/dp/1331480	0.3	1	0.3
2	VRef ajustable 2.495V a 36V 100mA TL431	http://uk.rs-online.com/web/p/voltage-reference/6619815/	0.15	1	0.15
3	Comparador IC único LM311P	http://es.farnell.com/texas-instruments/lm311p/ic-comparator-single/dp/1564937/	0.13	1	0.13
4	Módulo de puerto serie Bluetooth para Arduino JY-MCU	http://dx.com/p/jy-mcu-arduino-bluetooth-wireless-serial-port-module-104299	7.17	1	7.17
5	Inversor disparador Schmitt CMOS 14 PDIP	http://es.rs-online.com/web/p/inversores-trigger-schmitt/0526533/	0.36	1	0.36

Otro detalle a tener en cuenta a la hora de calcular el presupuesto consiste en considerar los costes de personal para el desarrollo del robot. En nuestro caso nos encontramos ante un primer prototipo que se ha creado desde 0, por lo tanto deberemos considerar las horas invertidas desde su concepción hasta su desarrollo final.

Han sido varias las personas que han colaborado o intervenido en el diseño y construcción del robot, por lo que para calcular los costes de personal consideraremos una estimación aproximada del tiempo invertido para cada una de las tres partes principales en las que se divide el proyecto.

Coste de personal		
Nº	Nombre/Descripción	Precio €
1	Parte electrónica	2600
2	Parte mecánica	2400
3	Parte informática	2200
	Total	7200

Una vez tenemos los distintos grupos de costes del robot, nos disponemos a realizar la suma de todos ellos para conocer su presupuesto final.

Presupuesto final		
Nº	Nombre/Descripción	Precio €
1	Software	0
2	Hardware	141.16
3	Otros elementos electrónicos	6.55
4	Plástico	0.38
5	Coste de personal	7200
	Total	7348.09

Finalmente incluimos al presupuesto final un 10% que corresponderían a los costes indirectos (tales como la luz, calefacción o el uso de materiales y equipos) que hayan podido aparecer durante el desarrollo.

Total	8082.9

9. Conclusiones

9.1. Conclusiones y trabajos futuros

Tras un largo proceso de desarrollo para dar forma a este proyecto, finalmente hemos podido finalizarlo. Ha sido un duro camino en el que se han tenido que sortear muchos problemas e inconvenientes, pero la colaboración mutua ha permitido superarlos.

Queríamos ser capaces de poder crear un robot móvil desde cero, empezando por su planteamiento y hasta su desarrollo y funcionalidad final.

Este proyecto comenzó a desarrollarse en el curso 2012-2013. Es cierto que en la fecha actual este robot no puede competir con otros robots de bajo coste que han surgido en estos últimos años. Nos encontramos ante una tecnología que está en constante evolución continua y que crece a pasos agigantados.

Sin embargo considerando la fecha en la que se planteó inicialmente, nuestro robot sí que habría cumplido ciertos objetivos. Entonces quisimos crear un robot con un coste inferior a los que se encontraban en ese momento en el mercado, y así lo hicimos.

Aunque se ha podido quedar limitado tecnológicamente, este robot es a día de hoy completamente operativo y funcional, y puede ser programado de múltiples maneras.

Para trabajos futuros podemos intentar recuperar este modelo de robot y rediseñarlo para desarrollar una versión más moderna y avanzada acorde con la actualidad.

Entre estas mejoras estaría la capacidad de ampliar el microcontrolador, ya que esto permitiría una mayor memoria y más capacidad a la hora de programar; con esto no nos encontraríamos con la falta de espacio que hemos tenido con el presente proyecto y podrían realizarse tareas más complejas con mayor libertad.

Incluir también una mayor cantidad de actuadores o sensores para disponer de una mayor libertad de movimientos y una mayor capacidad de información.

Este tipo de trabajos están en constante evolución, continuamente se pueden mejorar para incorporar una mayor funcionalidad y, esta es su mayor ventaja ya que están actualizándose continuamente.

10. BIBLIOGRAFÍA

1. INTRODUCCIÓN

- http://en.wikipedia.org

3. DISEÑO ELECTRÓNICO

3.1. Microcontrolador central

- http://www.frank-zhao.com/usnoobie/arduino.php

3.2. Gestión de energía

Tutoriales de la LIPO:

- http://www.youtube.com/watch?v=A6mKd5_-abk&feature=youtu.be

- http://www.rchelicopterfun.com/rc-lipo-batteries.html

Carga de baterías mediante alimentación USB:

- http://www.maxim-ic.com/app-notes/index.mvp/id/3241

Carga de baterías 2S mediante MCP73834:

- http://jameskbeard.com/Rowan/Electronics_I_ECE_Materials/Audio_Amplifier.pdf

Para 1s: Convertidor con UVLO programable:

- http://www.sparkfun.com/products/10255

3.3. Comunicaciones

Kit de adaptador XBee de Adafruit:

- https://www.adafruit.com/products/126

Configuración de los XBees con comandos AT:

- http://www.ladyada.net/make/xbee/configure.html

Programación Arduino sin cables:

- http://www.faludi.com/itp_coursework/meshnetworking/XBee/XBee_program_Arduino_wireless.html

Programación Arduino sin cables/link serie:

- http://www.ladyada.net/make/xbee/arduino.html

3.4. Sensores

- http://iteadstudio.com/store/index.php?main_page=product_info&cPath=4&products_id=415

Leer varios sensores infrarrojos con un único pin analógico

- http://webdelcire.com/wordpress/archives/1253

4. DISEÑO MECÁNICO

- http://iearobotics.com/oomlwiki/doku.php?id=start
- http://en.wikibooks.org/wiki/OpenSCAD_User_Manual
- http://asrob.uc3m.es/index.php/Impresora-3D_Open_Source
- http://es.wikipedia.org/wiki/Impresora_3D

5. DISEÑO SOFTWARE

- http://www.arduino.cc/

ANEXOS

1. Planos

1.1. Introducción

A continuación mostraremos todos los planos relacionados con los distintos elementos del robot y que han sido diseñados para este proyecto. Se pueden distinguir tres tipos de planos distintos:

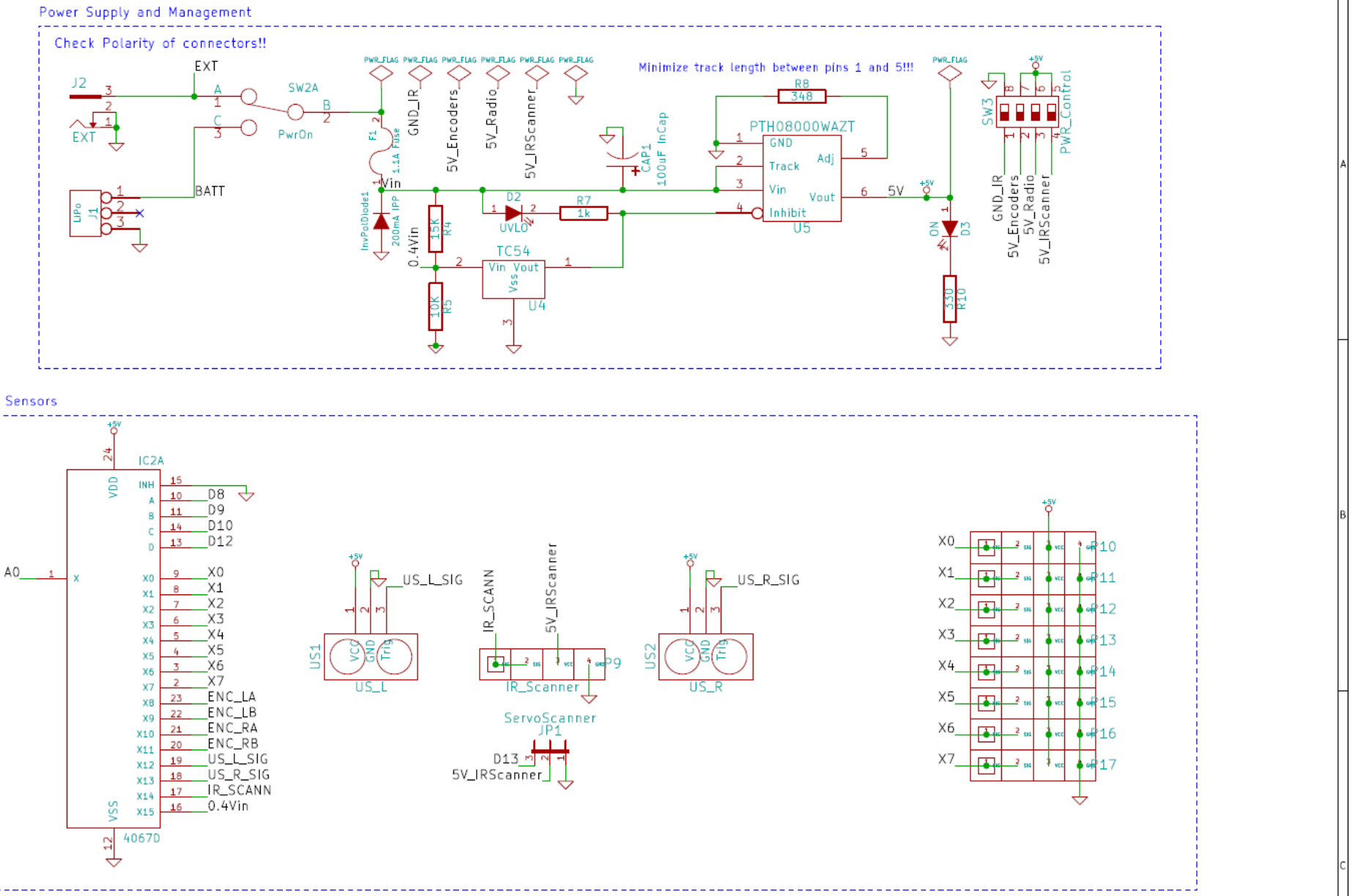
- Planos referidos al circuito electrónico.
- Planos referidos a la placa base.
- Planos referidos al chasis.

Los planos referidos al circuito electrónico nos muestran las conexiones entre los distintos componentes. Corresponde con el esquema electrónico del robot. Este esquema se ha diseñado con la ayuda del programa informático KiCad.

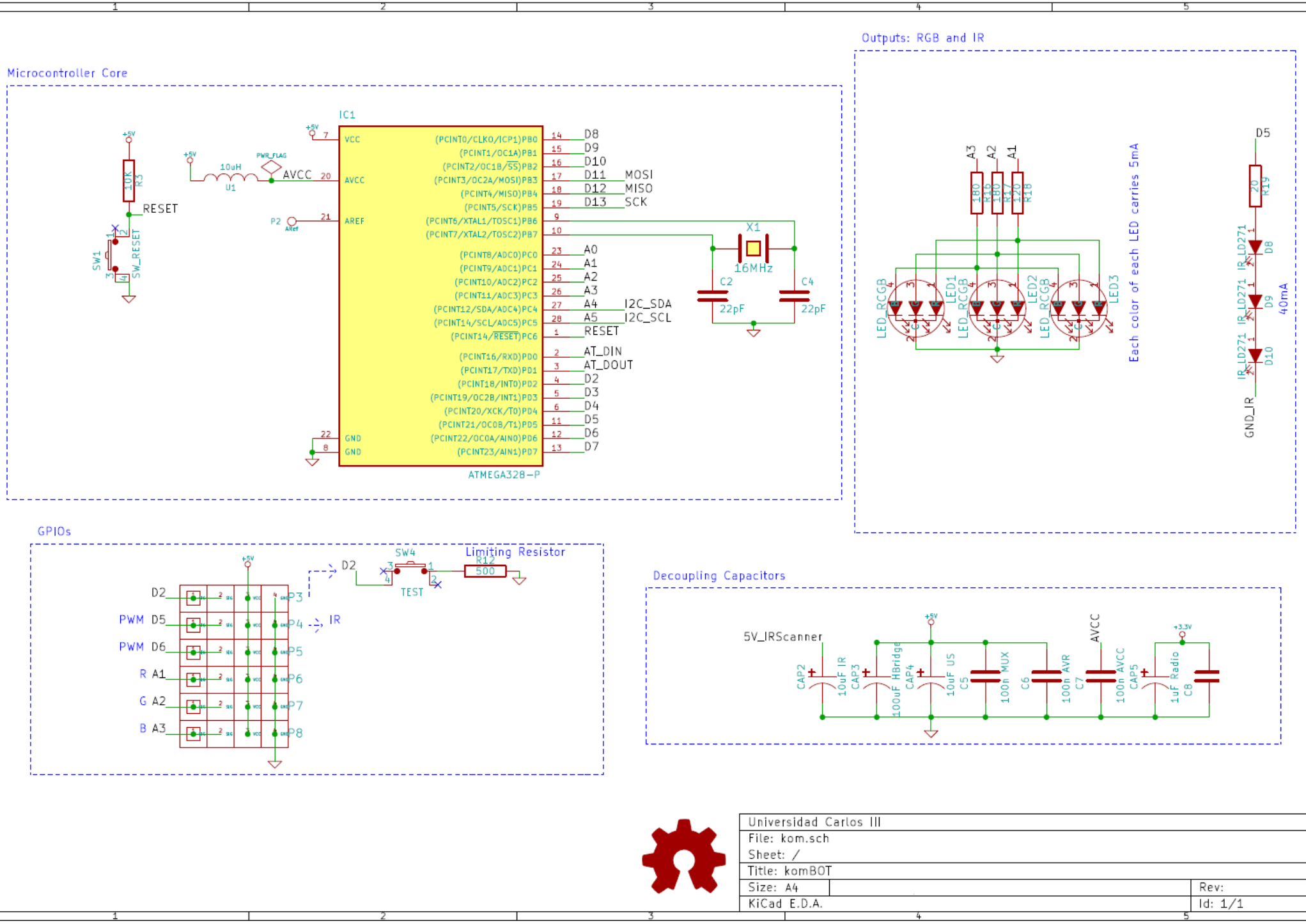
Los planos destinados a la placa base han sido realizados con la aplicación Pcbnew perteneciente al programa KiCad. Estos planos han sido acotados en milímetros.

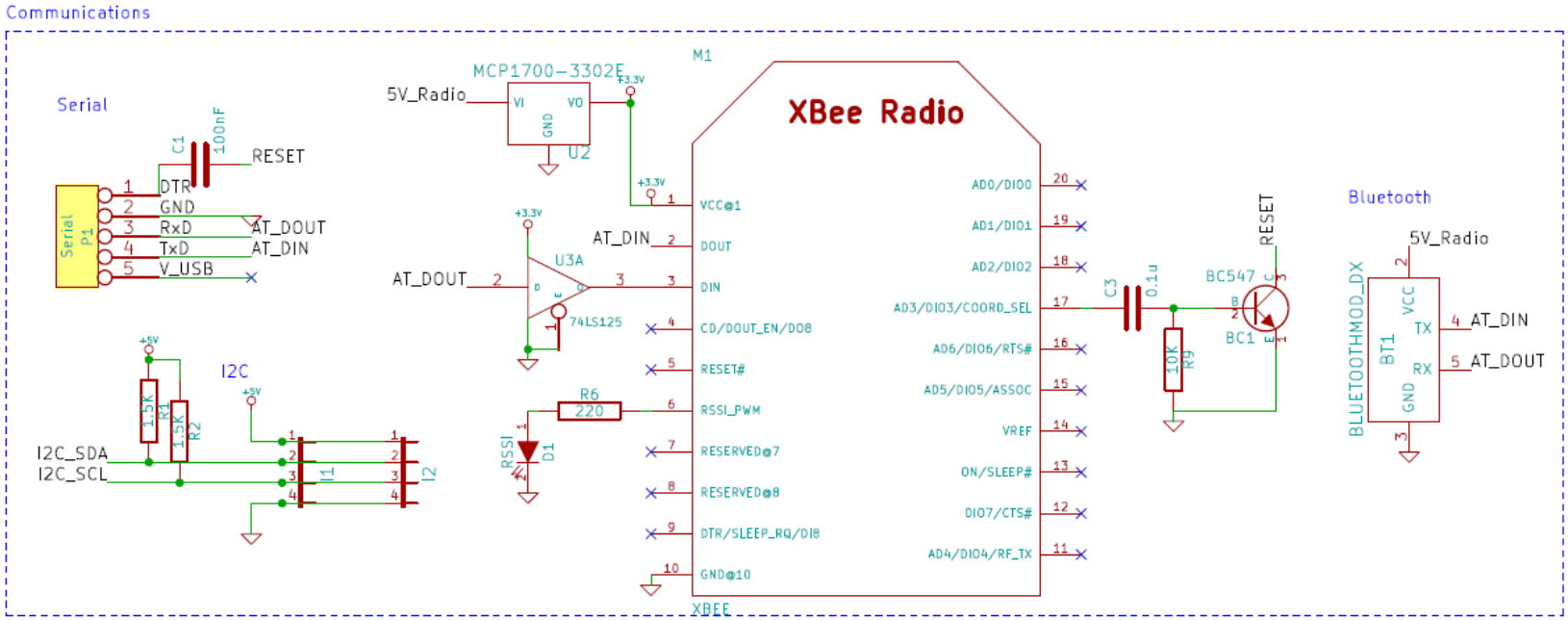
Los planos que muestran el diseño del chasis han sido creados con el programa de AutoCAD 2013 y las cotas que se muestran están referidas en milímetros.

1.2. Planos referidos al circuito electrónico



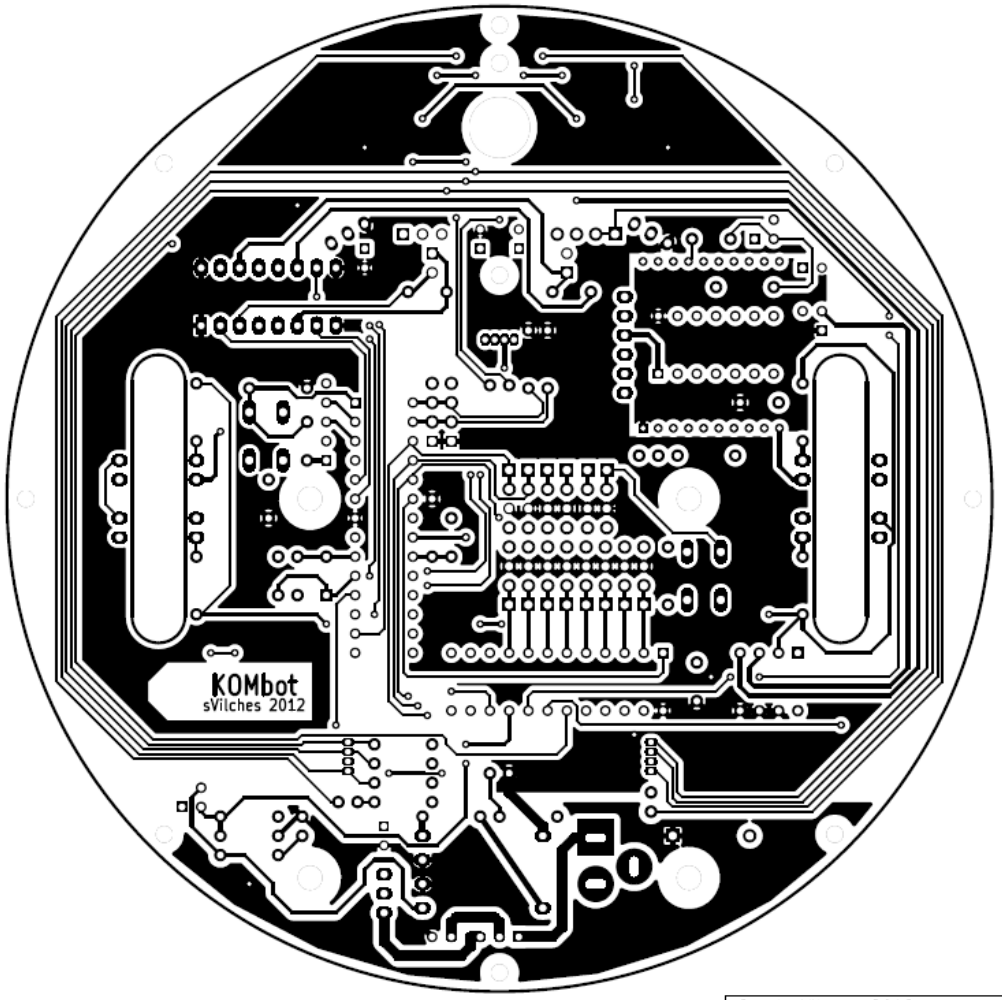
Universidad Carlos III		
File: kom.sch		
Sheet: /		
Title: komBOT		
Size: A4		Rev:
KiCad E.D.A.		Id: 1/1


Microcontroller Core
IC1
ATMEGA328-P
RESET
SW_RESET
10K
R3
10uH
U1
PWR_FLAG
AVCC
16MHz
X1
22pF
C2
C4
I2C_SDA
I2C_SCL
MOSI
MISO
SCK
AT_DIN
AT_DOUT
Outputs: RGB and IR
LED_RCGB
LED1
LED2
LED3
Each color of each LED carries 5mA
40mA
GND_IR
GPIOs
Limiting Resistor
TEST
IR
PWM D5
PWM D6
R A1
G A2
B A3
Decoupling Capacitors
5V_IRScanner
10uF IR
100uF HBridge
10uF US
100n MUX
100n AVR
100n AVCC
1uF Radio
Universidad Carlos III
File: kom.sch
Sheet: /
Title: komBOT
Size: A4
Rev:
KiCad E.D.A.
Id: 1/1

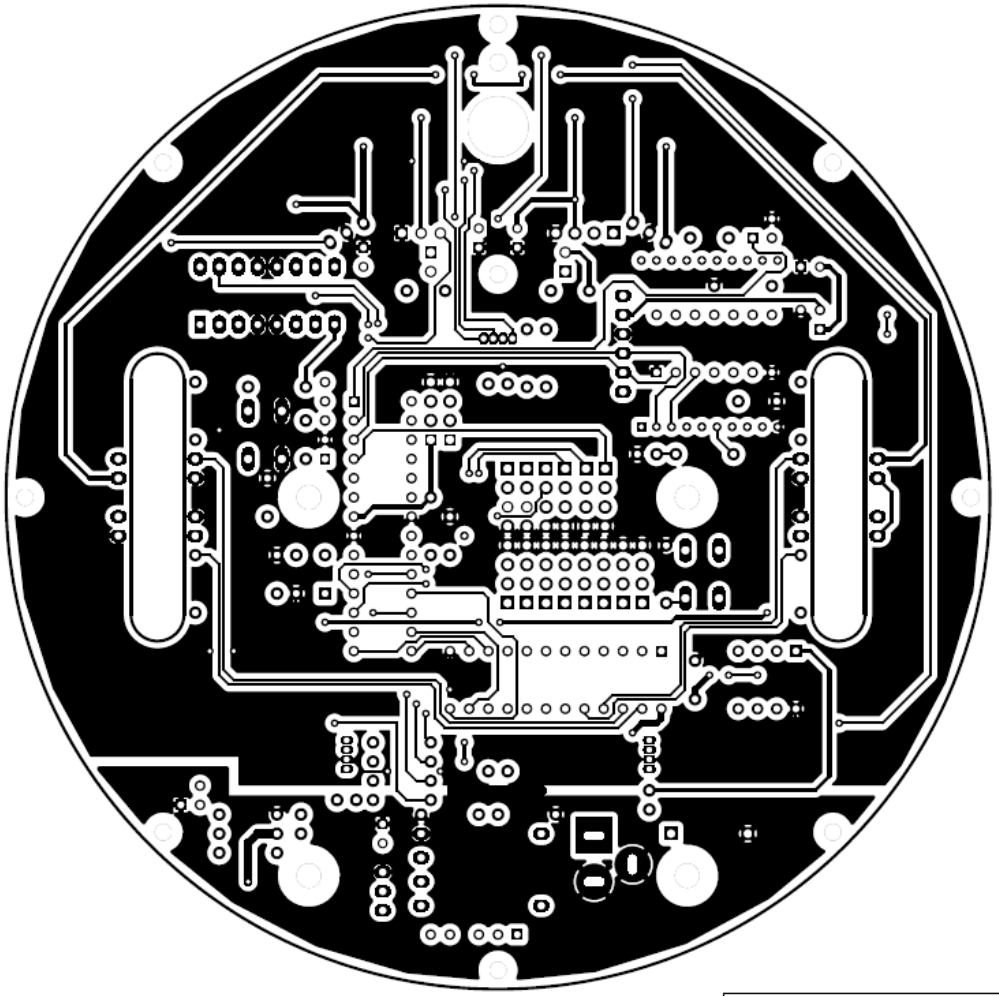


Universidad Carlos III	
File: kom.sch	
Sheet: /	
Title: komBOT	
Size: A4	Rev:
KiCad E.D.A.	Id: 1/1

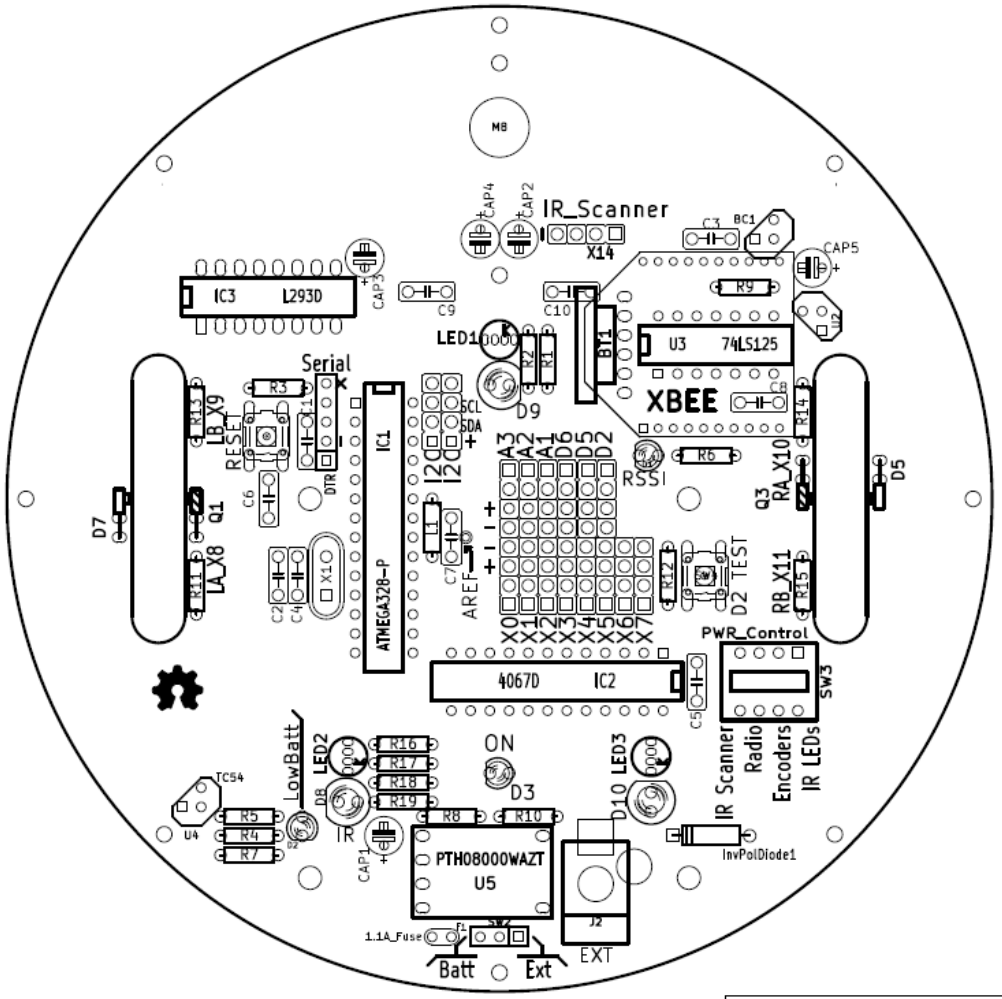
1.3. Planos referidos a la placa base



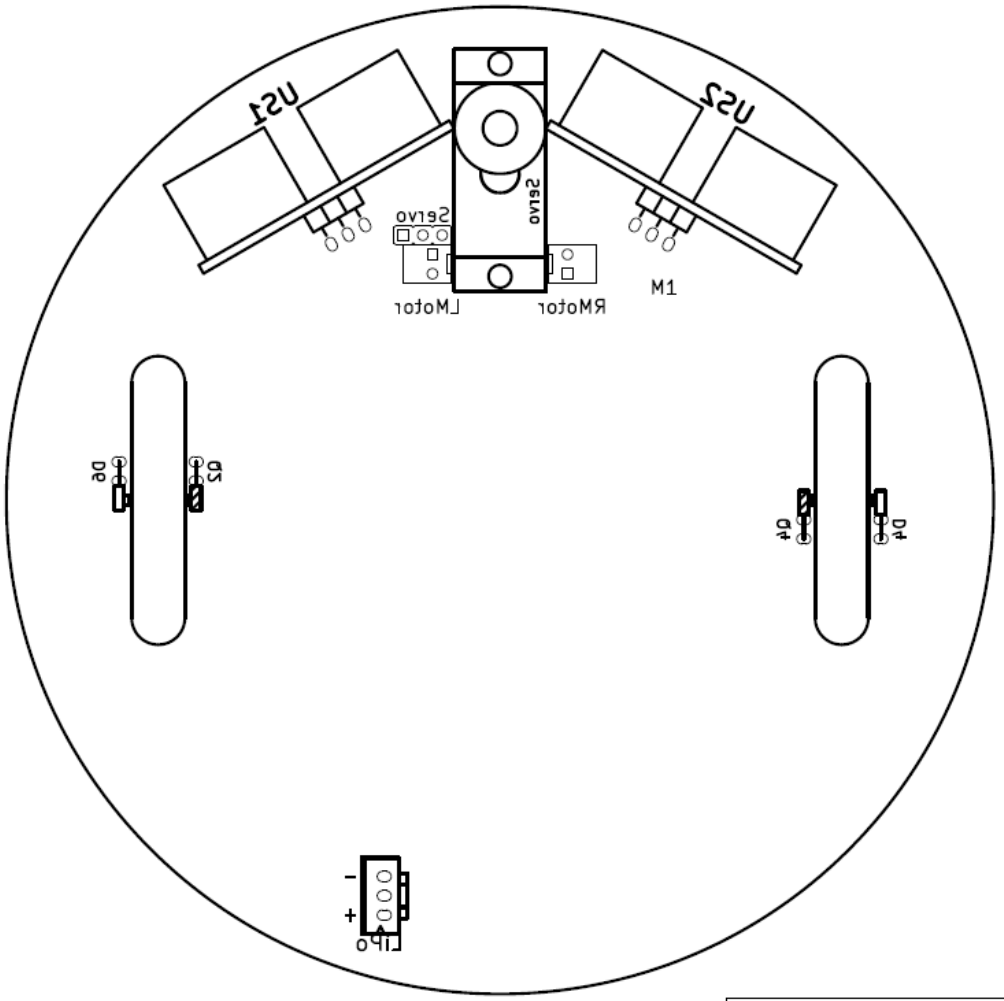
Sergio Vilches 2012 Creative Commons by-sa license PCB diameter: 130mm Universidad Carlos III de Madrid		
File: kom.brd Sheet: 1/1		
Title: KOMbot		
Size: A4		Rev: Alpha
KiCad E.D.A.		Id: 1/1



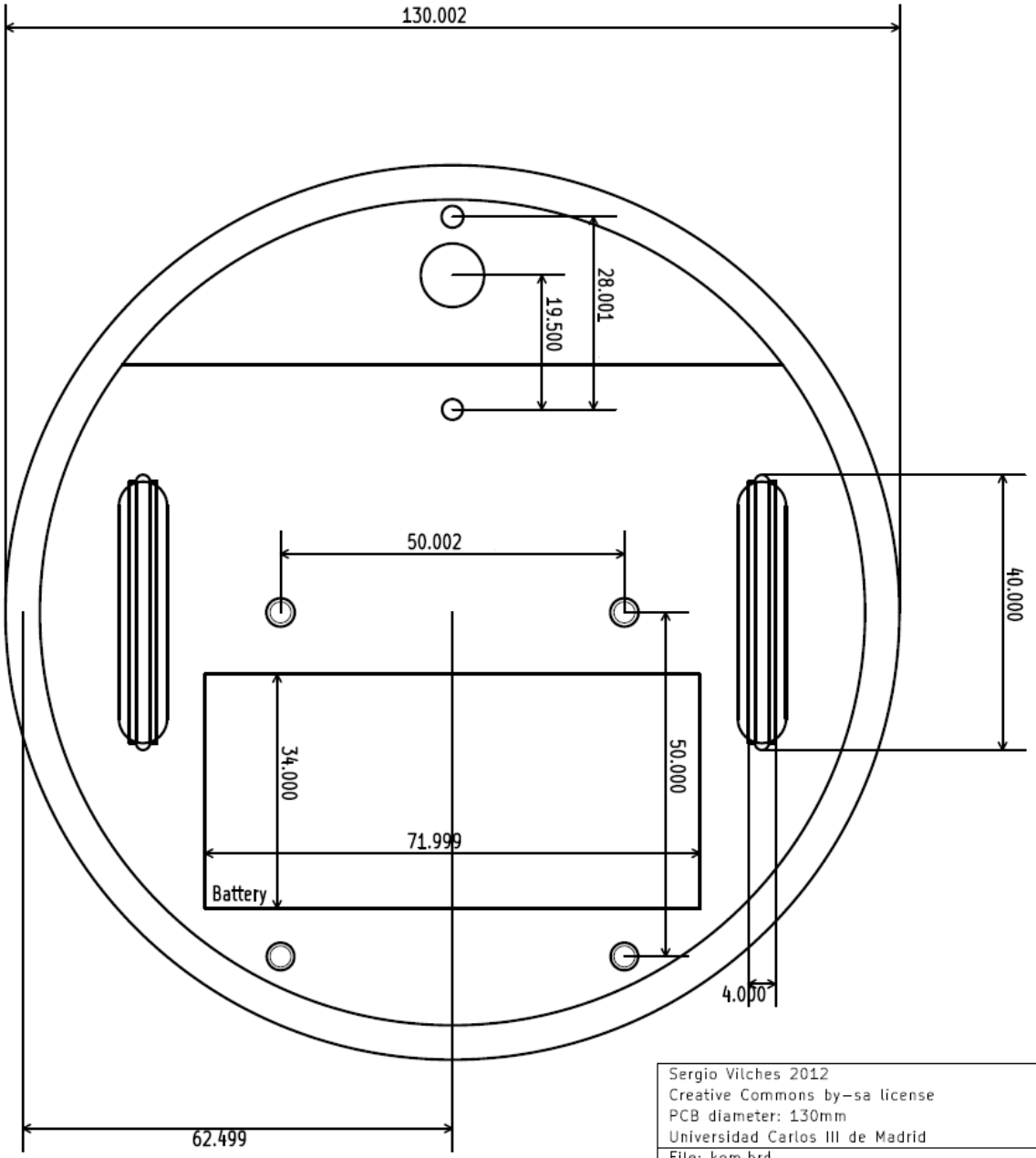
Sergio Vilches 2012 Creative Commons by-sa license PCB diameter: 130mm Universidad Carlos III de Madrid		
File: kom.brd Sheet: 1/1		
Title: KOMbot		
Size: A4		Rev: Alpha
KiCad E.D.A.		Id: 1/1



Sergio Vilches 2012		
Creative Commons by-sa license		
PCB diameter: 130mm		
Universidad Carlos III de Madrid		
File: kom.brd		
Sheet: 1/1		
Title: KOMbot		
Size: A4		Rev: Alpha
KiCad E.D.A.		Id: 1/1

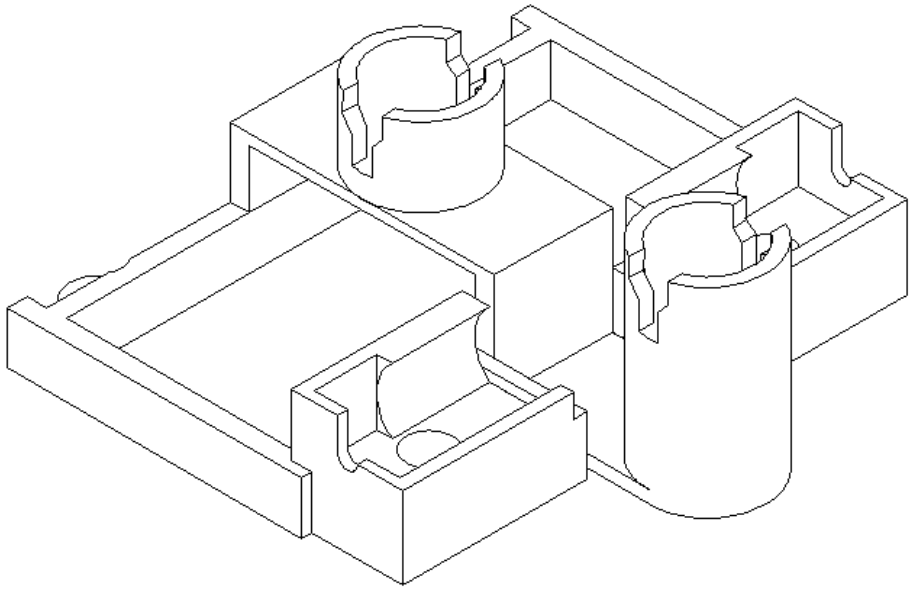


Sergio Vilches 2012 Creative Commons by-sa license PCB diameter: 130mm Universidad Carlos III de Madrid		
File: kom.brd Sheet: 1/1		
Title: KOMbot		
Size: A4		Rev: Alpha
KiCad E.D.A.		Id: 1/1

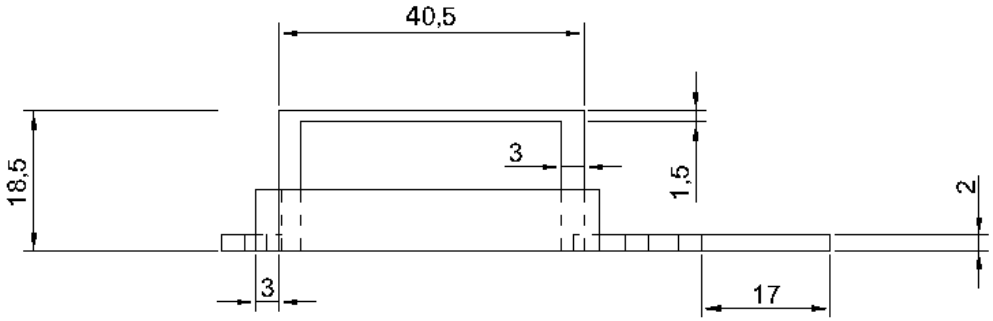


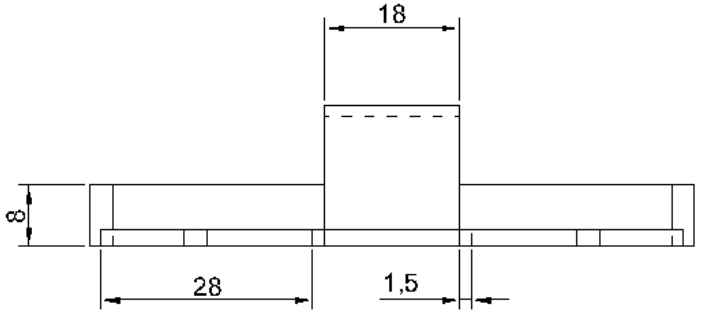
Sergio Vilches 2012 Creative Commons by-sa license PCB diameter: 130mm Universidad Carlos III de Madrid		
File: kom.brd Sheet: 1/1		
Title: KOMbot		
Size: A4	Date: 9 jul 2012	Rev: Alpha
KiCad E.D.A.		Id: 1/1

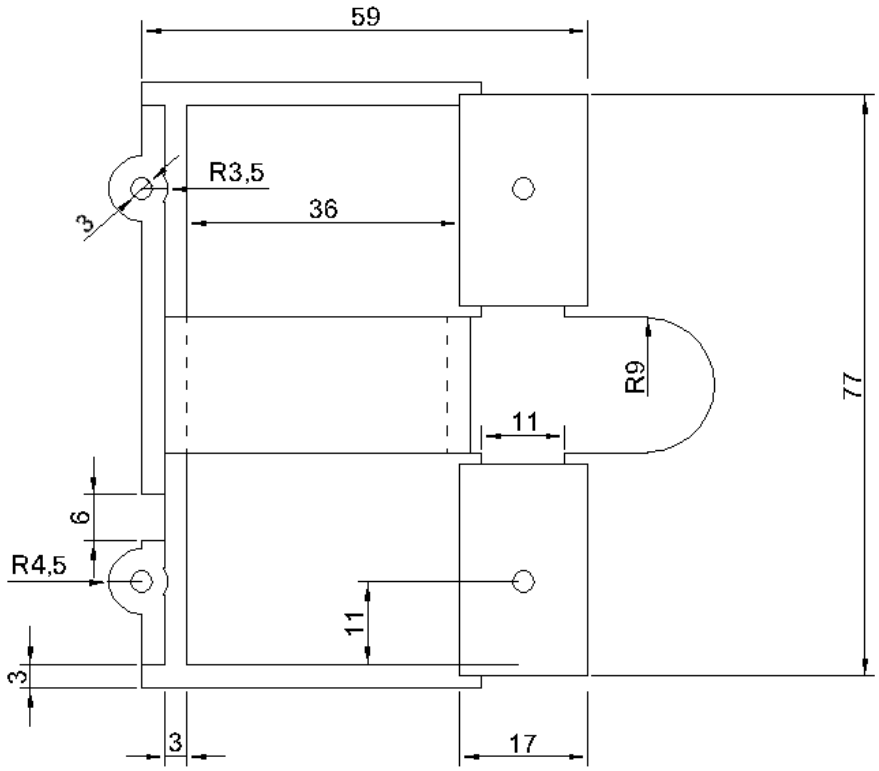
1.4. Planos referidos al chasis



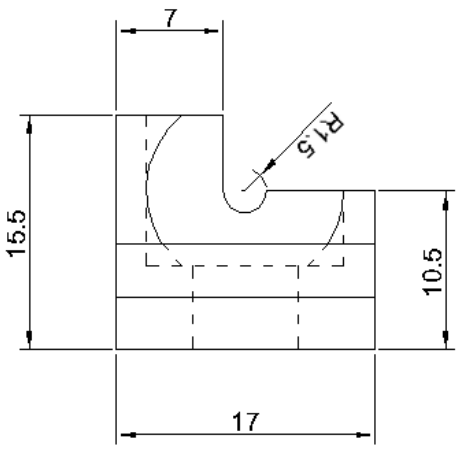
TÍTULO	
Kombot. Diseño y construcción de un mini robot móvil	
Plano Vista de isométrica del chasis	**Nº Plano** 0
Ingenieros D. Alberto Maldonado Sánchez	**Escala** 1:1
Propiedad Universidad Carlos III de Madrid	

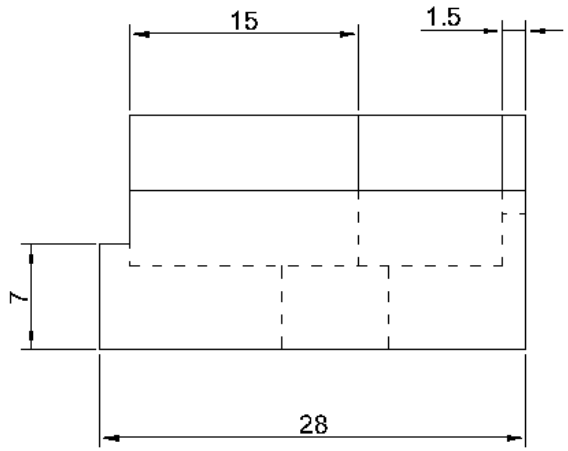


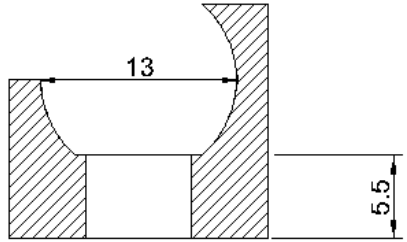


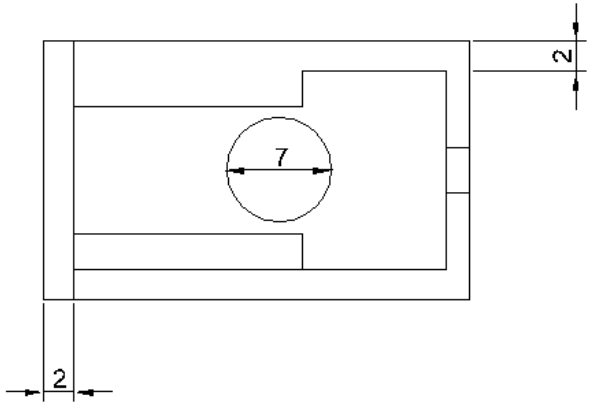


TÍTULO Kombot. Diseño y construcción de un mini robot móvil	
Plano Base del chasis	**Nº Plano** 1
Ingenieros D. Alberto Maldonado Sánchez	**Escala** 1:1
Propiedad Universidad Carlos III de Madrid	

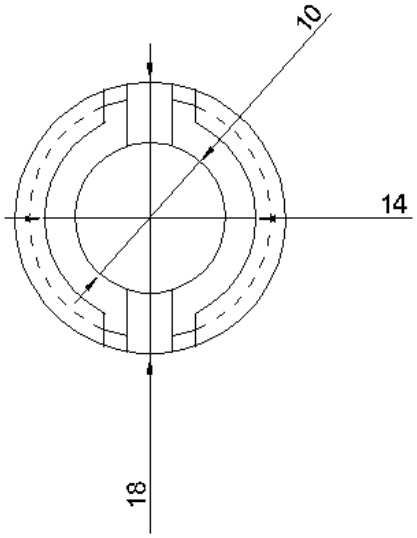


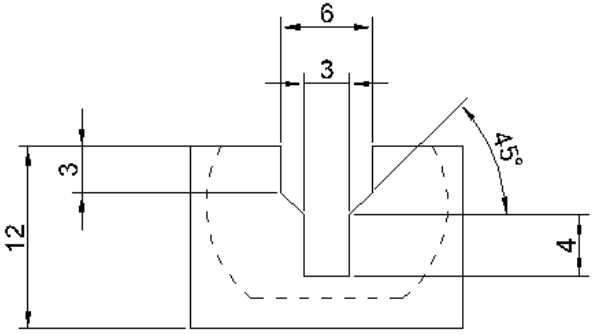


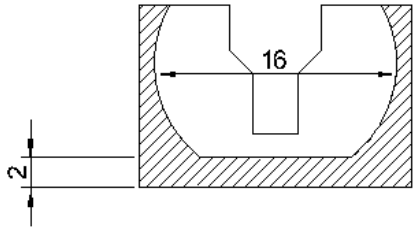




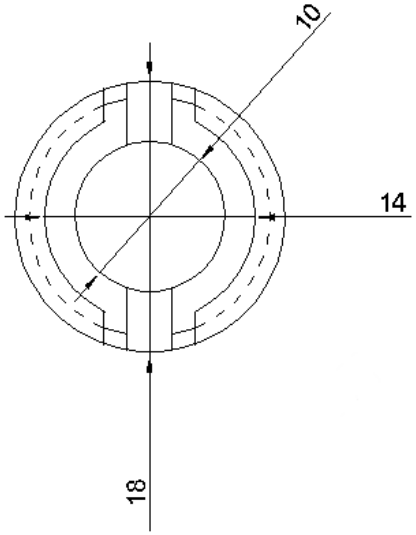
TÍTULO Kombot. Diseño y construcción de un mini robot móvil	
Plano Cavidad de los motores	**Nº Plano** 2
Ingenieros D. Alberto Maldonado Sánchez	**Escala** 2:1
Propiedad Universidad Carlos III de Madrid	

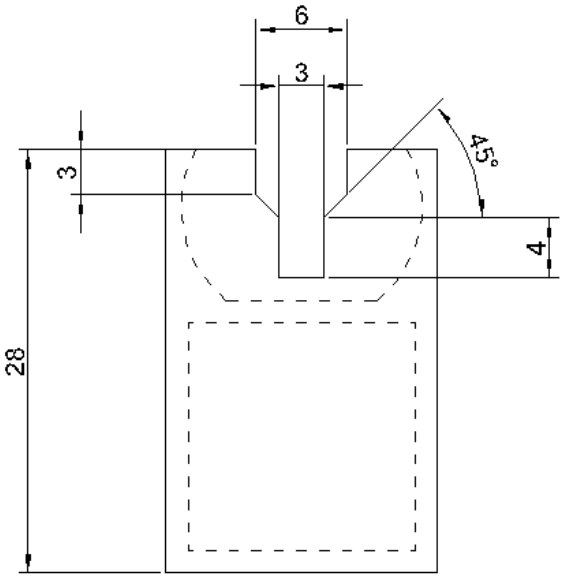


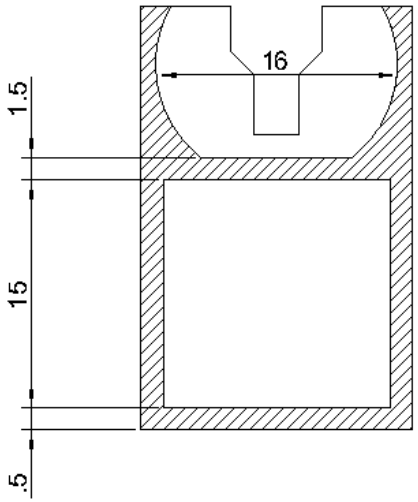




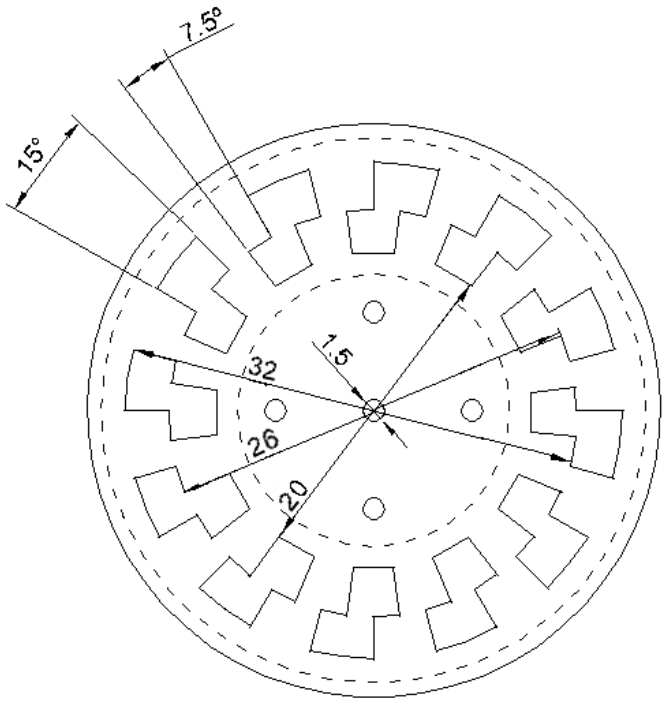
TÍTULO Kombot. Diseño y construcción de un mini robot móvil	
Plano Rueda canica trasera	**Nº Plano** 3
Ingenieros D. Alberto Maldonado Sánchez	**Escala** 2:1
Propiedad Universidad Carlos III de Madrid	

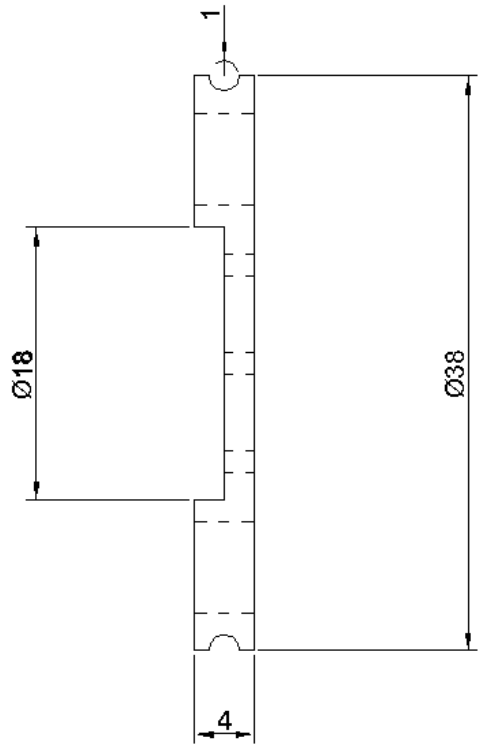






TÍTULO Kombot. Diseño y construcción de un mini robot móvil	
Plano Rueda canica delantera	**Nº Plano** 4
Ingenieros D. Alberto Maldonado Sánchez	**Escala** 2:1
Propiedad Universidad Carlos III de Madrid	





TÍTULO Kombot. Diseño y construcción de un mini robot móvil	
Plano Ruedas laterales	**Nº Plano** 5
Ingenieros D. Alberto Maldonado Sánchez	**Escala** 2:1
Propiedad Universidad Carlos III de Madrid	